PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito final para obtener el título de INGENIERO DE SISTEMAS

IMPLEMENTACIÓN DEL ALGORITMO DE COLONIA DE HORMIGAS PARA UN MOTOR DE JUEGOS SERIOS EMERGENTES

Por

Br. Andrés Roberto Trejo Sosa

Tutor: Dr. José Aguilar

Noviembre 2019

IMPLEMENTACION DEL ALGORITMO COLONIA DE HORMIGAS PARA UN MOTOR DE JUEGOS SERIOS EMERGENTES

Br. Andrés Roberto Trejo Sosa

Proyecto de Grado – Sistemas Computacionales – 45 páginas

Resumen: Un motor de juegos serios emergentes (MJSE) debe hacer explícito la posibilidad de emergencia en un Juego Serio, a partir del manejo coordinado de tramas de juegos, adaptadas al contexto educativo específico donde se esté desarrollando. Este proyecto se centra en la implementación del algoritmo de optimización de colonia de hormigas (ACO) para ser incluido dentro del MJSE de tal manera que se permita la emergencia de tramas según el tema que se está impartiendo dentro de un salón de clases inteligente. El MJSE realiza la gestión de un conjunto de tramas de juegos que pueden ser de interés en un contexto-dominio educativo, con el fin de adaptar el Juego Serio Emergente (JSE) inicialmente concebido al tema impartido en el salón de clases inteligente, y de esta forma emerger el Juego Serio adecuado al proceso pedagógico en curso. Adicionalmente, en este trabajo se realiza la comparación de la propuesta con otros trabajos recientes desde el punto de vista de que cualidades son o no implementadas dentro del MJSE.

Palabras claves: Motor de Juegos Serios Emergentes, Juegos Serios Emergentes, Algoritmo de Colonia de Hormigas, Salón de Clases Inteligente.

A mi madre Maira.

A mi hermana Adriana.

A la memoria de mi padre Roberto.

Índice

Índice de Figuras

Índice de Tablas

Agradecimientos

Primeramente, agradezco a mis padres Roberto Trejo y Maira Sosa, por ser pilares fundamentales de mi vida, ofreciéndome educación, valores y sustento. Apoyándome en todo momento, mediante ejemplo y esfuerzo.

Agradezco a mi padrino, David Márquez, por estar presente en todas las etapas de mi vida. Aportando sabiduría, consejos y motivación constante. Su ejemplo fue lo que me impulsó a escoger esta carrera.

Agradezco a la señora Amelia Pereira, quien me vio crecer, por brindarme cariño y apoyo incondicional durante toda mi vida.

Agradezco a mi familia, en especial a mi madrina Teresa Márquez y a mi tío Luis Alberto Sosa, por ser ejemplo de tenacidad e integridad y por siempre tenerme presente y brindarme su apoyo.

Agradezco a los profesores de la Universidad de los Andes, quienes me aportaron conocimientos, destrezas y habilidades en mi formación como profesional a lo largo de la carrera de Ingeniería de Sistemas, exigiéndome excelencia y calidad en cada paso.

En especial, agradezco a mi tutor, Prof. José Aguilar por su esfuerzo y dedicación, aportándome conocimientos, orientaciones y motivación durante todo el desarrollo de este proyecto de grado y servir de guía satisfactoriamente en esta área de estudio, hasta lograr culminar esta etapa importante de mi carrera.

Y para finalizar, agradezco al Ing. Francisco Díaz, por el apoyo y confianza que en conjunto con el Prof. José Aguilar, tuvieron en proponerme el desafío de desarrollar este proyecto de grado.

Capítulo 1

Introducción

En este trabajo se propone el desarrollo de un motor de Juegos Serios Emergentes (MJSE), concretamente el sub-motor de tramas, basado su implementación en el algoritmo de optimización Colonias de Hormigas (ACO, por sus siglas en inglés), los cuales permitirán la integración y adecuación autonómica de varios Juegos Serios en un MJSE, el cual sigue las dinámicas de aprendizaje de un Salón de Clases. En ese sentido, el MJSE posibilitará la adaptación de los Juegos Serios Emergentes (JSE) al aula, a partir de la emergencia de nuevos temas, reglas, estrategias, elementos, entre otros, de acuerdo a las necesidades de la clase. De esa manera, un JSE es visto como un objeto más de aprendizaje en el salón de clases que debe adaptarse, de forma que sea posible coadyuvar a mejorar los procesos de aprendizaje que ocurren en un salón de clases inteligente (SaCI).

1.1. Antecedentes

En [1] se define, explícitamente, el desarrollo y aplicación de un algoritmo ACO para la resolución del problema del viajante asimétrico, (ATSP, por sus siglas en inglés), cuyo fin es encontrar una solución que, satisfaciendo las condiciones iniciales del problema, proporcione una ruta o circuito cerrado cuya longitud sea la mínima. Para ello, se realizaron una serie de pruebas, a través de las cuales se obtuvieron resultados que, posteriormente, se evaluaron y valoraron mediante el uso de técnicas estadísticas. En [2], el objetivo principal del trabajo consiste en la comparación y fusión de motivos de las proteínas amiloideas, extraídas de la base de datos AMYPdb, denotadas como expresiones regulares usando las reglas PROSITE. El método de fusión de motivos utiliza el algoritmo de optimización combinatoria ACO, haciendo uso de los aminoácidos del primer motivo para construir el grafo donde las hormigas caminarán. En [3] se describe la aplicación de un algoritmo perteneciente a la metaheurística ACO (MAX-MIN Ant System) a un problema de optimización combinatoria. Además, se propone una técnica alternativa en la construcción de soluciones en algoritmos ACO, inspirada principalmente en otra metaheurística llamada Búsqueda Tabú. El problema a resolver se denomina Problema de Asignación Cuadrática (QAP, por sus siglas en inglés, Quadratic Assignment Problem); el cual es uno de los problemas de optimización combinatoria más difíciles de resolver en la práctica.

Según [4], los Juegos Serios representan la última innovación en los videojuegos, por ello, proponen un marco para la selección de motores de Juego Serios, caracterizando cinco elementos para la comparación de los mismos, los cuales son: Fidelidad Audiovisual y funcional, Composibilidad, Accesibilidad, Redes y Heterogeneidad. Además, en ese documento describen varios motores de juegos (Cry Engine, Source Engine, Unreal y Unity), adecuados para garantizar excelentes Juegos Serios. Por otro lado, [5] presenta el uso de los Juegos Serios para fines terapéuticos, utilizados en personas que presentan estrés postraumático, fobias y trastornos mentales. En particular, presentan Snow World, un videojuego para pacientes con quemaduras en

su cuerpo, a los cuales se les coloca un casco y se les presentan imágenes de la antítesis del fuego, esto es, el frio, representado por nieve, pingüinos, entre otros, con la finalidad de controlar el dolor o erradicarlo, mientras se realiza la curación del usuario.

En [6] se propone un framework para diseñar y desarrollar videojuegos, el cual se basa en la descomposición de un motor de juegos, lo que permite la reutilización de sus submotores. Los submotores en que se descompone el motor de juegos se separan en los sistemas de renderizado gráfico, de detección de colisiones, audio, objetos de juego y reglas. En [7] se diseña un motor de eventos, que permite seguir la dinámica del juego, transmitir comandos a la lógica del juego, enviar y recibir mensajes; todo esto llevándose a cabo según los eventos recibidos.

Las bases teóricas de los Juegos Emergentes han sido introducidas en [8]. Además, en ese trabajo se analizan el conjunto de reglas que conducen a estrategias de juego complejos, en los juegos de SimCity, Lincity y Los Sims. Por otro lado, en [9] proponen el Juego Emergente denominado Metrópolis, que parte de la premisa que las ciudades son auto-gestionadas a través decisiones tomadas en conjunto por sus habitantes (jugadores), sin que exista un habitante con un papel más importante. En Metrópolis, emergen patrones urbanísticos en la ciudad a partir de las decisiones que sus habitantes toman. Recientemente, en [10], se propone una extensión a Metrópolis, introduciendo mecanismos emergentes a la dinámica del juego, para que se adapte a sus jugadores.

1.2. Planteamiento del Problema

La metaheurística de ACO fue propuesta en [11] como un método para resolver problemas de optimización combinatorios. Los algoritmos de optimización basados en colonias de hormigas son parte de la rama de la inteligencia colectiva, este es el campo de investigación que estudia algoritmos inspirados en la observación del comportamiento de enjambres (swarms). Los algoritmos de inteligencia colectiva están compuestos de individuos simples que cooperan a través de la auto-organización, es decir, sin ninguna forma de control central sobre los miembros del enjambre.

Un Juego Emergente, es un juego con vida propia, el cual toma en cuenta lo que haga el jugador y al entorno, para responder. De esta manera, el guion no está escrito de antemano, la historia no es única, no hay un camino ni final ni definido. [12] señala que la programación emergente se aplica en la creación de videojuegos, utilizando inteligencia artificial, haciendo las aplicaciones cada vez más autónomas y autodidactas.

Por otro lado, [13], señala que los Juegos Serios son entornos donde los jugadores tienen objetivos y desafíos claros, no necesariamente vinculados con la victoria como meta final. Los juegos proporcionan un ambiente motivador, un contexto de entretenimiento y auto-fortalecimiento, para que los jugadores “aprendan haciendo" a través de sus propios errores, gracias a desafíos adecuados a su nivel de competencia y a una realimentación constante. El fin de los Juegos Serios es coadyuvar, motivar, educar, entrenar, etc., a los jugadores.

A pesar de la evidencia de la eficacia de los Juegos Serios en el ámbito educativo, algunos autores han encontrado obstáculos para su uso, como por ejemplo [14]: es difícil ajustar los horarios asignados a una materia con el tiempo dedicado al juego, o los contenidos de los juegos muchas veces no responden a las necesidades de las asignaturas, o son hechos para un contexto y objetivo preciso.

A su vez, las formas de emergencia en un juego se pueden expresar como [8, 10]: a) Por la aparición de nuevos comportamientos en el juego; b) Por la aparición de nuevas secuencias/escenarios, tramas o temáticas en los juegos; c) Por el surgimiento de nuevas propiedades en los objetos que los componen; d) Por la aparición de patrones que reflejan los resultados finales de los juegos; e) Por el surgimiento de modelos de negocios alrededor de los juegos. Ahora bien, los actuales motores de juego permiten algunos tipos de emergencia, con excepción de los casos a, b y c, los cuales deben ser diseñados de manera explícita por el diseñador del juego.

1.3. Objetivos

1.3.1 Objetivos Generales

Implementar el Algoritmo de Colonia de Hormigas para un MJSE, con el objetivo de permitir la aparición de nuevas secuencias o tramas en el juego.

1.3.2. Objetivos Específicos

- Especificar la arquitectura del algoritmo ACO para el MJSE, con el fin de permitir la aparición de nuevas secuencias o tramas en el juego.
- Implementar el algoritmo ACO en el MJSE.
- Realizar pruebas en el contexto de un salón de clases.

1.4. Justificación

La presente investigación se enfocará en el desarrollo de un MJSE orientado a la educación, pues se considera que si en un salón de clases se va a utilizar videojuegos continuamente, se requiere que se adapten dinámicamente a las necesidades de aprendizaje del tema en un momento dado. En particular, un SaCI requiere de Juegos Serios que no sean hechos para un contexto y objetivo específico, que se puedan adaptar a los requerimientos que vayan surgiendo en el aula, y en especial, la emergencia de nuevas tramas adecuadas a las temáticas que se impartan en el SaCI.

1.5. Alcance

Al finalizar el proyecto, se contará con la especificación detallada del submotor de tramas encargado de la aparición de nuevas secuencias o tramas en el juego, cuyo núcleo está definido por un algoritmo ACO. Además, también se tendrá su integración en el MJSE, el cual lo invocará para adaptar los JSE a las necesidades del tema de la clase que se esté impartiendo. Por otro lado, se elaborará un prototipo sobre la plataforma de MJSE, para evaluar su impacto en un salón de clases.

1.6. Metodología

1.6.1. Descripción de la metodología RAD (Rapid Application Development)

Rapid Application Development (RAD, por sus siglas en inglés) es un modelo de desarrollo ágil enfocado principalmente en la rápida creación de prototipos de un producto de software, realizándose iteraciones frecuentes basadas en retroalimentación, y publicando continuamente versiones actualizadas de dicho producto al mercado.

1.6.2. Fases de la Metodología

En general, la metodología RAD cuenta con cuatro fases principales:

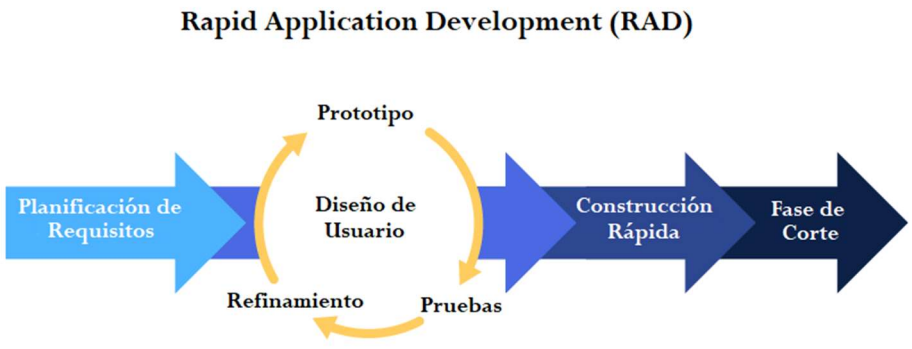

Figura 1.1 Fases de la Metodología RAD (tomado de [15])

1.6.3. Descripción de las Fases de la Metodología

La metodología está compuesta por las siguientes fases [16]:

Fase I: Planificación de Requisitos: Esta fase es equivalente a una reunión de alcance. A pesar de que la fase de planificación es mucho más corta en comparación con otro tipo de metodologías, sigue siendo un paso crítico para el éxito del proyecto. Durante esta etapa, los desarrolladores, clientes (usuarios del software), y los miembros del equipo se comunican para determinar las metas y expectativas del proyecto; como también, los problemas actuales y potenciales que tienen que ser identificados durante el desarrollo del producto.

- Una descomposición básica de esta etapa implica:
- Investigación del problema actual.
- Definición de los requerimientos del proyecto.
- Finalización de los requerimientos, con la aprobación del cliente.

Es importante que todos los participantes tengan la oportunidad de evaluar y dar su opinión sobre las metas y expectativas del proyecto. Al tener la aprobación de cada cliente y desarrollador, el equipo puede evitar problemas relacionados con falta de comunicación y cambios de decisiones, que a la larga pueden significar costos muy elevados durante el proceso de desarrollo.

Fase II: Diseño de Usuario: El diseño comprende los fundamentos básicos de toda metodología RAD. Durante esta fase, los clientes trabajan en conjunto con los desarrolladores, para asegurar que las metas se cumplan en cada paso del proceso de diseño.

Todos los errores son resueltos a través de un proceso iterativo: el desarrollador diseña un prototipo, el cliente lo prueba, y luego se llega a un acuerdo de lo que funciona y lo que no. Esta forma de trabajo le da la oportunidad a los desarrolladores de retocar el modelo durante todo el proceso de construcción, hasta lograr un diseño satisfactorio.

Fase III: Construcción Rápida: La fase 3 toma los prototipos obtenidos en la fase de diseño y los convierte en el modelo de trabajo. Debido a que la mayoría de los problemas y los cambios fueron identificados durante la exhaustiva fase de diseño, los desarrolladores pueden construir el modelo de trabajo final de una manera mucho más rápida, en contraste con lo generado al aplicar un enfoque de gestión de proyectos tradicional.

La fase se descompone de la siguiente manera:

- Preparación para el proceso de construcción rápida.
- Desarrollo de programas y aplicaciones.
- Codificación.
- Pruebas unitarias, de integración y de sistema.

El equipo de desarrollo de software trabaja en conjunto durante esta etapa, para asegurar que todo esté funcionando correctamente y que el resultado final satisface las expectativas y los objetivos del cliente. Esta

tercera fase es importante debido a que el cliente aún tiene la oportunidad de opinar durante el proceso de desarrollo, sugiriendo alteraciones, cambios, e incluso, nuevas ideas que pueden llegar a resolver problemas de forma inmediata.

Fase IV: Fase de Corte: Esta es la fase donde el producto final se libera al mercado. Incluye procesos de conversión de datos, pruebas y entrenamiento de usuarios. Todos los cambios finales son realizados, mientras desarrolladores y clientes continúan buscando errores en el sistema.

1.7. Organización de la Tesis

Este trabajo se organiza a través de seis capítulos a mencionar:

En el Capítulo 1 se introducen los conceptos básicos del problema, se describen los antecedentes necesarios para fundamentar este proyecto, así como se exponen los objetivos generales y específicos. Finalmente, se presenta la metodología y el alcance del proyecto.

En el Capítulo 2 se describen las bases teóricas que sirven como apoyo del proyecto. Se presentan una serie de conceptos relacionados con los JSE, se especifica la arquitectura detrás del funcionamiento de un MJSE, y por último, la definición de ACO.

En el Capítulo 3 se presenta el diseño del Módulo de Recuperación de Trazas y el Modulo de Emergencia de Secuencias. Se especifica la arquitectura del software junto con las fórmulas utilizadas en cada componente, que en conjunto, representan toda la funcionalidad del sistema.

El Capítulo 4 se enfoca en todo lo relacionado con las pruebas realizadas sobre el sistema. Se describen los objetivos de las pruebas, las métricas utilizadas, los resultados obtenidos, y las conclusiones generadas a partir de cada escenario.

Posteriormente, en el Capítulo 5 se realiza una comparación de cualidades entre el proyecto y otros trabajos que tienen relación con el tema desarrollado.

Finalmente, en el Capítulo 6 se presentan algunas conclusiones acerca de las funcionalidades y el desempeño de los métodos propuestos junto con algunas ideas para realizar investigaciones posteriores.

Capítulo 2

Marco Teórico

2.1. Juegos Serios Emergentes

En [4, 17, 18] definen a los Juegos Serios como juegos diseñados y desarrollados desde un objetivo distinto a la pura diversión. Los Juegos Serios proveen un ambiente motivador, un contexto de entretenimiento y auto-fortalecimiento, para que los jugadores "aprendan haciendo" a través de sus propios errores, gracias a desafíos adecuados a su nivel de competencia y a una realimentación constante.

Por otro lado, [8] define a un Juego Emergente como uno que se va desplegando de manera espontánea, autónoma, y sin leyes explícitas, adecuándose a los jugadores. En [9] proponen el Juego Emergente "Metrópolis", cuyo funcionalismo parte de la premisa que las ciudades generadas dentro del juego son auto-gestionadas por decisiones tomadas en conjunto por sus habitantes (jugadores), sin que exista un habitante con un papel más importante. Concluyentemente, en Metrópolis emergen patrones urbanísticos en la ciudad por las decisiones que sus habitantes toman. [10] propone una extensión a Metrópolis con la incorporación de mecanismos emergentes que permiten adaptar sus propiedades a la dinámica del juego introducida por sus jugadores.

A su vez, un JSE posiciona al jugador en un entorno de realimentación de información y motivación al logro, guiado por un objetivo explícito distinto de la pura diversión, para superar desafíos adecuados a su capacidad, y aprender de sus propios errores. En específico, el comportamiento que va dándose en el JSE resulta espontáneo, autónomo, y sin leyes explícitas, adecuándose a los jugadores y a sus entornos. En un JSE, la historia, la dinámica, el guion que va surgiendo, depende del contexto donde se va dando el juego.

Según [12], los tipos de emergencia que se pueden dar en un JSE son los siguientes:

Estrategias: se generan nuevas logísticas (serie de acciones encaminadas hacia un fin determinado) y tácticas (procedimiento o método que se siguen para ejecutar algo), siguiendo las normas, leyes y reglas del videojuego. Estas emergencias no han sido diseñadas, creadas, ni predefinidas por el diseñador del juego; por ejemplo, la emergencia de estrategias de golpes, tácticas de combos de ataque, etc. en videojuegos de combate.

Secuencia/Trama: se crean nuevas tramas (orden cronológico de los acontecimientos presentados) o temáticas (contexto de su desarrollo) en los juegos, lo que puede implicar cambiar el ambiente del juego, los eventos que aparecen en su dinámica, entre otras cosas. Por ejemplo: cambio de escenarios o de época en juegos tipo "Los Sims".

Propiedad: cambia las características y capacidades en los objetos, lo que puede conllevar a nuevos escenarios, personajes, etc. Eso puede implicar el cambio de normas, leyes y reglas en el videojuego, por ejemplo: jugar en sentido de las agujas del reloj en el dominó.

Final: determina cuando debe terminar el videojuego. Algunas cosas que podrían definirse en este tipo de emergencia son: hacer emerger vidas infinitas, finalizar el juego cuando se alcance un objetivo, entre otras cosas. Por ejemplo, en el juego "Metrópolis" [10], al aparecer ciertos patrones de interés (patrones urbanísticos), se podría dar por terminado el juego.

Modelo de Negocio: según [19], tiene que ver con el surgimiento de modelos de servicios alrededor de los juegos. Por ejemplo, en algunos juegos aparece un sistema de comercio para comprar e intercambiar personajes, herramientas, entre otras cosas, como es el caso de "Top Gear" con la compra de cauchos, de motor, etc.

Utilidad: hace emerger como se va a utilizar el JSE, en función del contexto o la narrativa del ambiente donde se usa. Por ejemplo: "Era Mitológica" puede ser utilizado para explicar hechos históricos, geográficos o religiosos.

2.2. Emergencia de Secuencia/Tramas

Las tramas se refieren al orden cronológico de diversos acontecimientos presentados a un jugador en un juego. Dichas tramas están vinculadas a las narrativas (gameplay) que determinan las escenas en el mundo del juego. Así, la trama realiza la unión del hilo histórico de una narrativa. En este sentido, es un concepto referido al conjunto de acontecimientos de una historia según el orden causal y temporal en el que ocurren los hechos. En los videojuegos, la "historia" son secuencias animadas llamadas "escenas", con o sin diálogo entre personajes no jugables (NPC), mientras el jugador va controlando al Avatar o protagonista del juego. Por otro lado, la "mecánica" del juego son reglas, procesos y datos, que definen cómo progresa un juego, qué sucede cuando se gana o se pierde, entre otras cosas. En un juego de computadora, están incorporadas en el código del juego [20]. Por otro lado, según [21], la narrativa es el contexto que se establece alrededor de la mecánica en el juego, y se visualiza mediante "escenas", donde el jugador no tiene control de ellas. A continuación, describiremos los tipos de narrativas clásicos en los videojuegos:

Narrativa lineal: es como en una película, tiene un comienzo, un medio y un final predeterminados. Entonces, la linealidad existe en un juego desde que el jugador determina cuando comienza, qué sucede mientras está jugando, y cómo termina.

Narrativa adaptada: no es lineal, no es planificado por el programador, como por ejemplo: comprar productos dentro del juego, usar el juego en otro contexto, no terminar en el tiempo planificado.

Narrativa emergente: "aparece" cuando el jugador interactúa con el juego y "desaparece" cuando el jugador deja de interactuar con el juego. Emerge de historias no creadas por el programador, cada vez que el jugador realiza algún tipo de interacción. Es, una narrativa no lineal.

2.3. Arquitectura de un MJSE

En [22] definen a los motores de juegos como los ambientes computacionales que permiten realizar videojuegos. Pueden ser vistos como programas, librerías o frameworks, para el desarrollo de videojuegos. Un motor de videojuegos es el núcleo general que une todas las partes de un juego. Así, los desarrolladores se centran en las mecánicas, las lógicas y las características específicas del juego que está concibiendo.

Por otro lado, en un JSE, el MJSE maneja un conjunto de tramas, y las va seleccionando y fusionando en una única trama, según el contexto y el objetivo del juego. Para ello, el MJSE requiere realizar el siguiente conjunto de tareas:

- La búsqueda y selección del conjunto de tramas adecuadas al contexto donde funcionará el JSE.
- La fusión de algunas de las tramas seleccionadas en una única trama, según los objetivos del Juego Serio, que se desplegará inicialmente en el videojuego, y la supervisión de la dinámica del desarrollo del JSE, para adecuarlo a la dinámica del contexto donde funciona, a través de la adaptación de sus tramas.

La arquitectura del MJSE que se presenta está basada en [23, 24]. En dichos trabajos, el Motor se encuentra dividido en capas jerárquicas, cuyos componentes son presentados en las siguientes subsecciones.

2.3.1. Núcleo del Motor de Videojuego (NMV)

Es el elemento central del MJSE, en él se encuentran los seis submotores de base para cualquier videojuego, los cuales son (ver Figura 2.1):

Submotor de gráficos (SG): realiza y maneja los gráficos, imágenes y dibujos primitivos, basados en sus características: texturización, mallas, terrenos, etc.

Submotor físico (SF): se encarga de realizar los movimientos físicos de los objetos en un ambiente virtual.

Submotor de sonido (SS): se encarga de gestionar todo lo referente a lo audible: música, audio, ruido, micrófonos, entre otras cosas.

Submotor de interacción (SI): se encarga de configurar las interacciones dentro y fuera de los videojuegos.

Submotor de video (SV): se encarga de la unión del sonido y las imágenes en secuencias fílmicas para realizar videoclips, caricaturas, cortes de películas, etc., usadas en el videojuego.

Submotor de renderización (SR): se encarga de gestionar las imágenes en movimiento, considerando aspectos como: mejorar la iluminación, sombreado y oscuridad, definir sus efectos visuales, entre otros.

Figura 2.1 Arquitectura del Motor de Juegos Serios Emergentes

2.3.2. Subsistemas de Emergencia del Videojuego (SEV) y de Adaptación del Videojuego (SAV)

Son las capas del MJSE que permiten hacer emerger un JSE [23]. En específico, ambos subsistemas usan los siguientes submotores:

Submotor de IA (SIA): se encarga de introducir comportamientos inteligentes en los diferentes componentes del JSE. Para ello, en este componente se despliegan las diferentes técnicas usadas de la IA para permitir la emergencia en un JSE. Un ejemplo de ello es el algoritmo ACO, utilizado por el Submotor de Trama Emergente (STE) cuando es invocado por el SEV, para definir la primera versión del JSE.

Submotor de Trama Emergente (STE): es el responsable de hacer emerger en el JSE las narrativas y las secuencias de las tramas adaptadas al contexto. Para ello, recolecta la información del contexto, realiza la gestión de escenas y eventos, ensambla subtramas/guiones de diferentes juegos, entre otras cosas.

2.3.2.1. Subsistema de Emergencia del Videojuego (SEV)

En el caso del STE, permite hacer emerger la primera versión del JSE que será ejecutada, según los objetivos que se deben cumplir. El STE, cuando es invocado por el SEV, está compuesto por los siguientes componentes (ver [23, 24], para más detalles del STE):

Gestor de Materia (GM): determina la temática que se está tratando en el contexto para, a partir de allí, establecer el objetivo que debe cubrir el JSE.

Gestor de Videojuegos (GV): busca en repositorios de videojuegos (por ejemplo, edugame, advergame, etc.), subtramas o videojuegos para una temática en particular. Dichas subtramas/videojuegos son definidos como Recursos de Aprendizaje (RA). Para la búsqueda, se realiza una comparación entre los metadatos de los RAs en los repositorios, y la temática definida por el GM. Si no consigue al menos un videojuego parecido a lo buscado, se hace un llamado al módulo siguiente.

Módulo de Generación de JSE (MGJSE): es el responsable del ensamblaje de un nuevo JSE usando las subtramas provistas por el GV. En un trabajo previo, se ha diseñado un MGJSE basado en ACO [3]. El MGJSE tiene imbricadas las funciones de los siguientes tres componentes del SEV, para generar inicialmente un JSE.

Storyboard (SB): se encarga de generar los guiones narrativos o subtramas del JSE.

Gestor de Escenas (GE): genera el ambiente, mundo o entorno, requerido por las tramas del JSE.

Sistemas de Eventos (SE): se encarga de generar eventos especializados requeridos por el SB, para generar los comportamientos deseados en el JSE.

2.3.2.2. Subsistema de Adaptación del Videojuego (SAV)

En el caso del STE, permite ir adaptando al JSE durante el desarrollo del mismo. La capa SAV permite que en un JSE se generen comportamientos emergentes durante el juego, actuando sobre sus características de base. En particular, esta capa permite la adaptación de las características de sus elementos, la emergencia de nuevas estrategias, secuencias de tramas, ambientes y eventos, en el videojuego. En [12] se proponen 6 niveles de emergencia en un Juego Emergente, que se dividen en dos módulos, a saber:

Módulo de Emergencia Fuerte: este módulo está compuesto de tres subcapas que permiten la emergencia fuerte en el videojuego, de la siguiente forma:

- **Estrategias**: se generan nuevas tácticas y logística en el juego, siguiendo reglas con variantes.
- **Secuencia**: se crean nuevos escenarios, tramas o temáticas en los juegos, cambiando el ambiente y el contexto del juego.
- **Propiedad**: cambia las características en los objetos.
- **Módulo de Emergencia Débil**: este módulo está compuesto de tres subcapas que permiten la emergencia débil en el videojuego, de la siguiente forma:
- **Final**: patrones que reflejan resultados de los juegos, o continuar el juego de forma infinita (en el caso de los Juegos Serios no es necesario que termine el juego, ni que haya un ganador).
- **Modelo de Negocio**: por el surgimiento de mercados y servicios alrededor de los juegos.
- **Utilidad**: depende para que se va a utilizar el JSE.

2.4. Algoritmo de Colonia de Hormigas (ACO)

ACO es un tipo de meta heurística basada en una población, el cual está inspirado en la conducta de las colonias de hormigas reales cuando buscan comida. Ha sido utilizado para resolver problemas de optimización combinatoria como se describe en [24, 25, 26]. ACO está compuesto por:

Espacio de solución: es un grafo o espacio que recorrerán las hormigas para obtener soluciones.

Hormigas: caminan en el grafo.

Solución: los nodos del grafo son marcados por una feromona, igual que los arcos que los interconectan. Cuando converge el algoritmo, los nodos son seleccionados según si su feromona pasa un umbral, igual que los arcos que salen de ellos, como parte de la solución final.

Feromonas: define lo deseable de los nodos y de los arcos que los interconectan, para pertenecer a la solución final.

Función Feromona: actualiza cada tipo de feromona, en función de la calidad de la solución propuesta.

Función Heurística: define la decisión heurística que toma una hormiga al estar en un nodo, con respecto a que otro nodo debe continuar a visitar después de él.

El macroalgoritmo clásico de ACO [24, 25, 26] se muestra a continuación:

Inicializar parámetros, feromonas y grafo
Repita Mientras (*no se cumpla terminación*)
    ConstruirSolucionesPorHormigas
    ActualizarFeromona
ConstruirSoluciónFinal

En general, las hormigas van recorriendo el grafo, y en la medida que lo recorren van construyendo una solución posible al problema estudiado. Durante el recorrido, ellos van tomando la decisión de qué nodo visitar según una función heurística que considera la cantidad de feromona depositada en el entorno y el objetivo que se busca alcanzar en el problema. Finalmente, cada hormiga deposita feromona en el recorrido realizado, según la calidad de la solución alcanzada.

Matemáticamente, el macroalgoritmo ACO usa las siguientes ecuaciones. La hormiga k se mueve del estado x al estado y según la siguiente probabilidad:

$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(n_{xy}^{\beta})}{\sum_{u \in J_x^k}(\tau_{xu}^{\alpha})(n_{xu}^{\beta})} \; si \; y \in J_x^k$$

Donde τ_{xy} es la cantidad de feromonas que se han depositado en la transición del estado x a y, $0 \leq \alpha$ es un parámetro para controlar la influencia de τ_{xy}, n_{xy} es la conveniencia de la transición del estado $x\ a\ y$, $\beta \geq 1$ es un parámetro para controlar la influencia de n_{xy}, y J_x^k es el conjunto de sitios que puede visitar la hormiga k desde la posición x.

Cuando todas las hormigas han completado una solución, los rastros son actualizados por:

$$\tau_{xy} = (1 - \rho)\tau_{xy} + \sum_k \Delta\tau_{xy}^k$$

Donde τ_{xy} es la cantidad de feromonas depositadas para la transición del estado $x\ a\ y$, ρ es el coeficiente de evaporación de feromonas y $\Delta\tau_{xy}^k$ es la cantidad de feromonas depositadas por la hormiga k.

Capítulo 3

Diseño

En este capítulo se presentan todos los aspectos del diseño e implementación del Submotor de Trama Emergente (STE) basado en el algoritmo ACO. En particular, se explicará cada uno de los componentes principales del Módulo de Recuperación de Trazas y del Módulo de Emergencia de Secuencias. El proceso de desarrollo se llevó a cabo en su totalidad haciendo uso del lenguaje de programación Java, en su versión 1.8.0. En la figura 3.1 se muestra de manera general, el flujo y los diferentes componentes del STE.

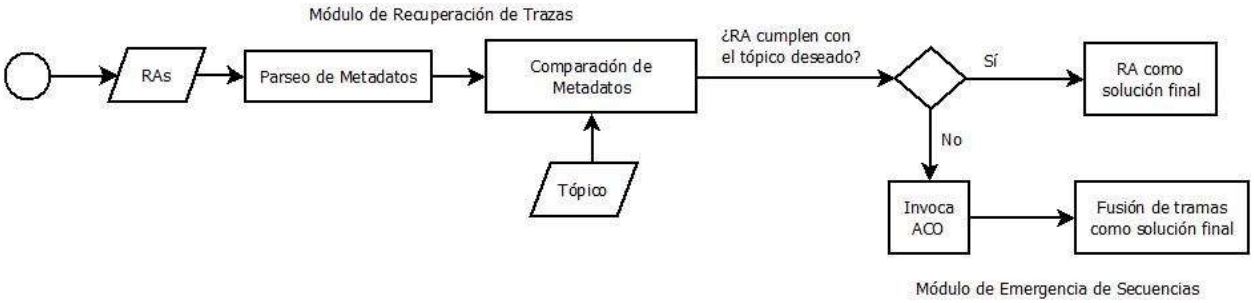

Figura 3.1 Diagrama de flujo del STE

3.1. Diseño del Módulo de Recuperación de Trazas

El Módulo de Recuperación de Trazas está compuesto por dos componentes principales: el componente de parseo de metadatos y el componente de comparación. Ambos componentes cumplen con un objetivo en común, el procesamiento de los recursos de aprendizaje (RA), para definir qué tan parecido es el contenido de los RA del tipo Juegos Serios con respecto a la temática que se esté buscando.

Cada RA está caracterizado por un metadato que sigue el estándar Learning Object Metadata (LOM) [27]. Dicho archivo (usualmente codificado en XML) tiene la finalidad de describir a través de una serie de atributos la temática de cada Juego Serio (RA). A continuación, se hace una descripción de los atributos tomados en cuenta durante el proceso de parseo (ver Figura 3.2):

Title: Es el nombre del RA.

Language: Es el lenguaje para el que fue hecho el RA.

Description: Contiene una descripción del contenido del RA. Por ejemplo: "Juego de dominó interactivo que relaciona sus piezas según las diferentes representaciones fraccionarias gráficas o numéricas".

Keyword: Palabras claves que representan el tema principal del RA. Por ejemplo: fracciones, dominó, cálculo, probabilidad.

Coverage: Describe la zona geográfica o región en la que es aplicable el RA.

Format: Identifica el software necesario para utilizar el RA.

TypicalAgeRange: Edad intelectual del destinatario típico del RA. Por ejemplo: "7-9", "0-5", "15".

Difficulty: Este elemento describe lo difícil que resulta el uso del RA para los usuarios típicos. Por ejemplo: muy difícil, difícil, medio, fácil, muy fácil.

Duration: Tiempo aproximado o típico para asimilar el RA.

InteractivityLevel: El grado de interactividad que caracteriza a ese RA. Se mide como, muy alto, alto, medio, bajo, muy bajo.

SemanticDensity: La densidad semántica de un RA puede ser estimada en función de su tamaño, ámbito, o en el caso de recursos auto-regulados, tales como audio y video, su duración. La densidad semántica de un RA es independiente de su dificultad. Se mide como: muy alto, alto, medio, bajo, muy bajo.

IntendedEndUserRole: Usuario(s) principal(es) para el(los) que ha sido diseñado el RA.

Context: El entorno principal para el que fue diseñado el RA.

CognitiveProcess: Es el tipo específico del proceso cognitivo del RA.

Cost: Indica si el RA requiere pago para su uso.

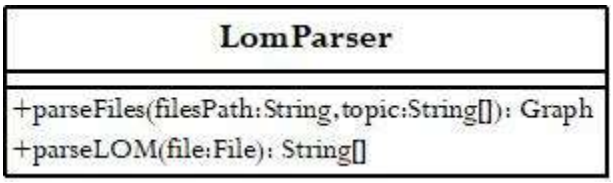

Figura 3.2 Diagrama de la clase LomParser

En nuestro caso, las tramas de Juegos Serios es lo que se buscará como RA.

Una vez realizado el proceso de parseo, en donde se utilizan los metadatos LOM de los RA para extraer los 15 atributos previamente descritos, es necesario comparar la información de cada atributo de la trama (RA) con respecto a la información del curso establecida (tópico que se desea impartir específicamente en el curso). Eso determina el interés/parecido de dicha trama (RA) con respecto al curso a través de un valor numérico. Durante el proceso de comparación se hacen uso de los siguientes criterios (ver Figura 3.3):

- Para el atributo "Title" se comparan las palabras que conforman al título del tópico deseado con las palabras del título del RA. Su valor determina la cercanía de dicha comparación, se devuelve 1 para un resultado correcto y 0 para un resultado incorrecto.

- Para el atributo "Language" se compara el lenguaje del tema deseado con el lenguaje del RA. Si son iguales, el valor obtenido es 1, si son diferentes 0.

- Para el atributo "Description" se comparan las palabras que corresponden a la descripción del tópico deseado con cada una de las palabras que conforman la descripción del RA. El valor obtenido corresponde al mayor entre todas las comparaciones, en un rango [0,1].

- Para el atributo "Keyword" se compara un conjunto de palabras clave separadas por coma, tanto del tópico deseado como del RA. Su valor determina la cercanía de dicha comparación, se devuelve 1 para un resultado correcto y 0 para un resultado incorrecto.

- Para el atributo "Coverage" se compara la cobertura del tema deseado con la cobertura del RA. Si son iguales el valor obtenido es 1, si son diferentes 0.

- Para el atributo "Format" se comparan un conjunto de formatos separados por coma, tanto del tópico deseado como del RA. Si alguno de los formatos coincide, el valor obtenido es 1, de lo contrario es 0.

- Para el atributo "TypicalAgeRange" se compara la edad especificada en el tópico deseado con la edad especificada en el RA. Si son iguales el valor devuelto es 1, si hay una diferencia de +/- 3 el valor devuelto es 0.5, de lo contrario el valor devuelto es 0.

- Para el atributo "Difficulty" se compara la dificultad especificada en el tópico deseado con la dificultad especificada en el RA. Usando una tabla como referencia, de acuerdo a la similitud entre las comparaciones, se devuelve un valor que puede ser 0, 0.25, 0.50, 0.75 o 1.

- Para el atributo "Duration" se compara la duración en minutos especificada en el tópico deseado con la duración en especificada en el recurso de aprendizaje. Si son iguales el valor devuelto es 1, si hay una diferencia de +/- 30 minutos el valor devuelto es 0.5, de lo contrario el valor devuelto es 0.

- Para el atributo "InteractivityLevel" se compara el nivel de interactividad especificado en el tópico deseado con el nivel de interactividad especificado en el RA. Usando una tabla como referencia, de acuerdo a la similitud entre las comparaciones, se devuelve un valor que puede ser 0, 0.25, 0.50, 0.75 o 1.

- Para el atributo "SemanticDensity" se compara la densidad semántica especificada en el tópico deseado con la densidad semántica especificada en el RA. Usando una tabla como referencia, de acuerdo a la similitud entre las comparaciones, se devuelve un valor que puede ser 0, 0.25, 0.50, 0.75 o 1.

- Para el atributo "IntendedEndUserRole" se comparan un conjunto de palabras separadas por coma, tanto del tópico deseado como del RA. Si alguna de las palabras coincide, el valor obtenido es 1, de lo contrario es 0.

- Para el atributo "Context" se comparan un conjunto de palabras separadas por coma, tanto del tópico deseado como del RA. Si alguna de las palabras coincide, el valor obtenido es 1, de lo contrario es 0.

- Para el atributo "CognitiveProcess" se comparan un conjunto de palabras separadas por coma, tanto del tópico deseado como del RA. Si alguna de las palabras coincide, el valor obtenido es 1, de lo contrario es 0.

- Para el atributo "Cost" se compara el costo del tema deseado con el costo del RA. Si son iguales el valor obtenido es 1, si son diferentes 0.

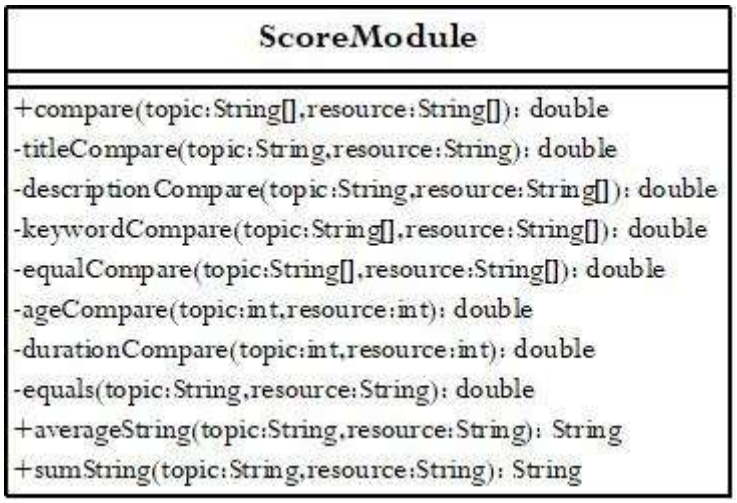

Figura 3.3 Diagrama de la clase ScoreModule

El promedio de los valores se convierte en el índice de similitud entre el RA y el tema deseado. Una vez que se realiza la comparación de todos los Juegos Serios con el tema deseado, el Modulo de Recuperación de Trazas ejecuta el siguiente algoritmo:

$$Si\ (Similitud(metadata\ RA_i, TemaDeseado) \geq 0.850)$$
$$entonces$$
$$Devuelve\ RA_i con\ esa\ puntuación\ como\ solución\ final$$
$$Si\ (0.400 \leq Similitud(metadata\ RA_i, TemaDeseado) \leq 0.850)$$
$$entonces$$
$$Invoca\ ACO$$

Ese algoritmo determina si hay algún Juego Serio que cumple con el tema deseado, de lo contrario, si consigue algunos RA (tramas) más o menos similares a la temática buscada, intenta hacer emerger un Juego Serio para el tema deseado usando ACO (ver sección 3.2).

3.2. Diseño del Módulo de Emergencia de Secuencias

El Modulo de Emergencia de Secuencias utiliza a ACO, que se encarga de crear agentes hormigas, con el fin de construir un JSE que cumpla con un objetivo específico (temática deseada). En particular, un videojuego seleccionado para formar parte del grafo de solución lo denominaremos subtrama, mientras que una trama de un JSE es la unión/fusión de una o más subtramas para generar una solución final. ACO está compuesto por:

Hormigas: son los agentes que caminan en el grafo de subtramas (ver Figura 3.4).

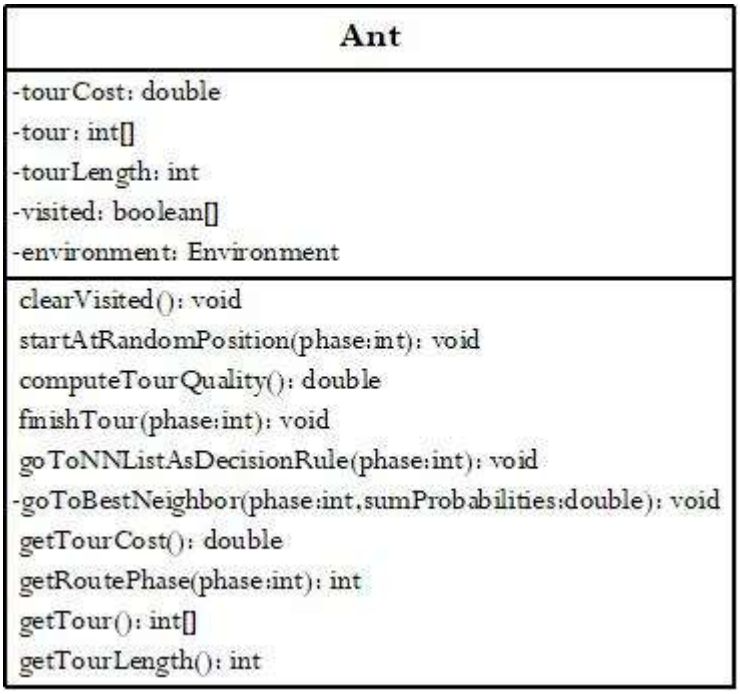

Figura 3.4 Diagrama de la clase Ant

Espacio de solución: espacio que recorrerán las hormigas para obtener soluciones. Es un grafo compuesto por nodos que definen las subtramas seleccionadas desde los diferentes repositorios, y arcos que establecen las relaciones de dependencia entre ellas, cuando existen. Los arcos establecen la secuencia lógica entre las subtramas. La figura 3.5 describe la clase que define el ambiente (grafo).

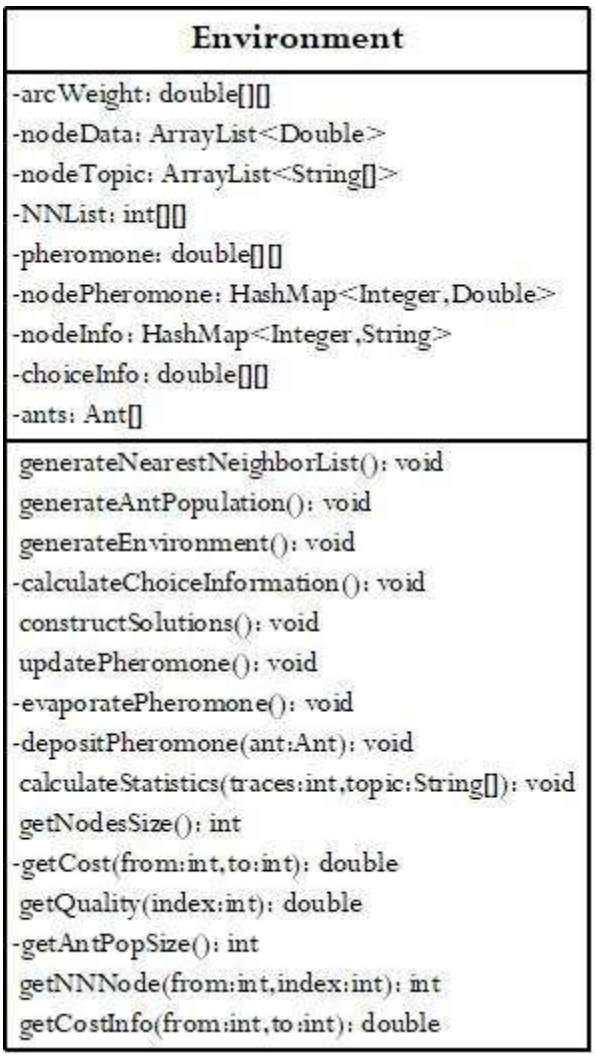

Figura 3.5 Diagrama de la clase Environment

Solución: las subtramas (nodos) son marcadas por una feromona en el grafo, igual que los arcos que los interconectan. Cuando converge el algoritmo ACO, las subtramas son seleccionadas según si su feromona pasa un umbral, igual que los arcos que salen de ellas, los cuales establecen la secuencia lógica de ejecución de las subtramas.

Feromonas: hay dos tipos, una para las subtramas (nodos) y otro para los arcos entre las subtramas. La feromona define lo deseable de la subtrama y de los arcos que las interconectan, para pertenecer a la solución final.

Función Feromona: actualiza cada tipo de feromona en función de la calidad del JSE propuesto.

Función Heurística: define la decisión que toma una hormiga, al estar en una subtrama (nodo), con respecto a que otra subtrama (nodo) debe continuar visitando desde ella.

ACO permite un proceso de aprendizaje colectivo entre las hormigas, para hacer emerger la nueva configuración del JSE. Así, la solución no es más que una secuencia de subtramas, ordenadas según una secuencia lógica entre ellas.

3.2.1. Macroalgoritmo ACO para el STE

El macroalgoritmo utilizado es el clásico de ACO [24]. En específico, en nuestro caso consiste de una fase de inicialización de parámetros, un proceso iterativo hasta que el sistema converja, y la construcción de la solución final. Dicho macroalgoritmo se detalla en las siguientes secciones:

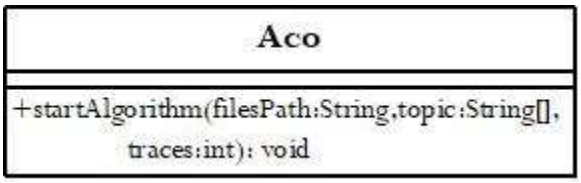

Figura 3.6 Diagrama de la clase Aco

3.2.1.1. Creación del grafo teórico de recorrido de las Hormigas

El grafo teórico es definido como $G = (N, E)$, donde N es un conjunto de nodos que representan las subtramas (Juegos Serios seleccionados), y E un conjunto de arcos que conectan todos los nodos de N (ver Figura 3.7). Por otro lado, se establece una función de peso d_{ij} para determinar el peso de un arco $(i, j) \in E$, tal que es 1.0 si existe una relación de dependencia secuencial entre dos nodos $(JS_i, JS_j, \in N)$, y 0 en caso contrario. Eso implica que $d_{ij} \neq 0$ cuando entre dos nodos hay una relación de dependencia entre ellos. El valor de peso luego es utilizado para agilizar la transición de las hormigas entre los nodos del grafo durante su recorrido.

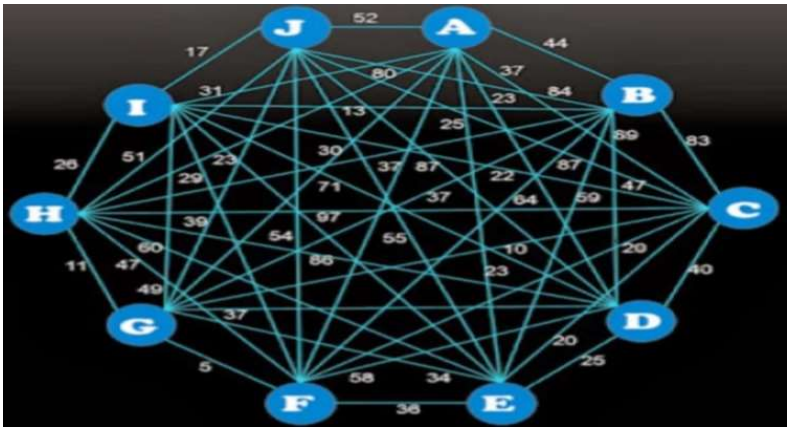

Figura 3.7 Grafo de recorrido de ACO

Los nodos del grafo guardan la información de la subtrama que representan (ID del nodo), pero adicionalmente, almacenan el nivel de similitud entre el nodo y la temática tratada (basado en sus metadatos, calculado por el Módulo de Recuperación de Trazas), y el nivel de feromona actualizado por las hormigas que transitan a través de ellos, que indica la deseabilidad de dicho nodo (ver Figura 3.8).

Figura 3.8 Nodo del grafo

3.2.1.2. Construcción de la solución por parte de las hormigas

En esta fase, algunas consideraciones se realizan:

1) Se define el número de hormigas que integran la colonia.
2) Se define un valor aleatorio de feromona inicial para cada arco.
3) Cada hormiga inicialmente se coloca de modo aleatorio en el grafo para iniciar su recorrido, y determina un JSE (una solución).

Cada hormiga ejecuta una función heurística (o de transición) desde el nodo actual donde se encuentra, para determinar el próximo nodo que visita que no haya previamente visitado. Esta función es definida como la probabilidad de visitar desde el nodo r a cada uno de sus nodos contiguos s, en función del nivel de feromona en el arco entre los nodos r y s $(\tau(r,s))$, pesado por el parámetro α que determina su influencia $(\alpha \geq 0)$, y del índice de similitud de cada nodo s $(\eta(s))$ con respecto a la temática buscada, el cual es pesado por otro parámetro β que determina su influencia $(\beta \geq 0)$. Esa función heurística se calcula para todos los nodos aun no visitados por la hormiga $k(J_r^k)$, usando la siguiente ecuación (1):

$$P_{(r,s)}^{k} = \frac{\tau_{(r,s)}^{\alpha} . n_{(s)}^{\beta}}{\sum_{u \in J_r^k} \tau_{(r,u)}^{\alpha} . n_{(u)}^{\beta}} Si\ s\ \in J_r^k$$

Ahora bien, una hormiga puede culminar en cualquier momento la construcción de una solución (JSE), o continuar paseando por los nodos, hasta recorrer a todos, basado en el siguiente algoritmo:

$$Recorrido\ Hormiga\ k = \begin{cases} parar, & num_{aleatorio} > umbral\_parar \\ continuar\ constr. JSE\ usando\ ec.(1), & caso\ contrario \end{cases}$$

3.2.1.3. Actualización de las Feromonas

En este caso hay dos feromonas, una para los arcos y otra para los nodos. Ambas son actualizadas al final de cada iteración (recorrido de cada hormiga). En ese sentido, cada hormiga actualiza la feromona de cada arista y de cada nodo que visita. Para ello, se determina un índice de la calidad del JSE propuesto por cada hormiga, el cual será usado durante el proceso de actualización. Ese índice de calidad es calculado a partir del nivel de similitud del JSE propuesto por cada hormiga, con respecto al tema deseado. El nivel de similitud del JSE propuesto por una hormiga k no es más que el índice de similitud "$\eta(s)$" derivado de la concatenación (fusión) de los atributos de las P subtramas que componen el JSE propuesto por la hormiga k (JSE^k), al compararlo con la temática buscada; a ese valor se le denominará $\Delta\tau^k$.

De esta manera, una vez finalizado su recorrido, la hormiga k realiza la actualización de feromona de cada arco $(\tau_{(r,s)})$ y nodo $(\tau_{(r)})$ usando las ecuaciones genéricas siguientes:

$$\tau_{(r,s)} = \tau_{(r,s)}(1-\rho) + \Delta\tau^k\ tal\ que\ (r,s)\ \in\ JSE^k$$

$$\tau_{(r)} = \tau_{(r)}(1-\rho) + \Delta\tau^k\ tal\ que\ r\ \in\ JSE^k$$

Donde, $\rho\ \in\ (0,1)$ es el coeficiente de evaporación de feromona. Para los arcos y nodos no recorridos por la hormiga k, sucede un proceso de evaporación de sus feromonas.

En general, este proceso se realiza repetidamente para todas las hormigas, hasta que la colonia converge en un grupo de soluciones (JSEs).

3.2.1.4. Construcción de la solución final

Una vez que la colonia concluye su trabajo, se debe pasar a construir la solución final, es decir, la nueva versión del JSE que propondrá ACO. Para ello, se hace un recorrido sobre todos los nodos del grafo, seleccionándose los nodos con mayor valor de feromona, y los arcos que saldrán de cada uno de ellos serán seleccionados según si los interconectan y su valor de feromona (serán los que tengan valor mayor), tal que se garantice que todos los nodos (subtramas) conformen un camino (esa será la secuencia lógica del nuevo JSE propuesto, ver Figura 3.9).

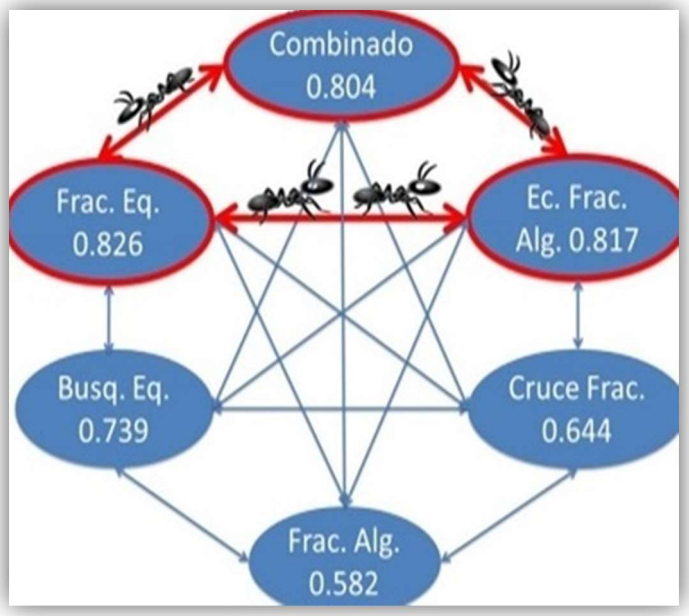

Figura 3.9 Nuevo JSE

Capítulo 4

Protocolo Experimental

En este capítulo se evalúa el funcionamiento del STE. El objetivo del STE es, una vez recolectada la información del entorno del SaCI, definir un JSE que se adapte a la temática actual del curso.

4.1. Contexto General Experimental

El repositorio Agrega fue escogido como la fuente de los RA a utilizar en el experimento (ver Figura 4.1). La federación de repositorios de objetos digitales educativos **Agrega** (ver, http:\\agrega.educacion.es) es una plataforma con contenidos educativos que se pueden descargar y usar directamente. Para el desarrollo de los distintos escenarios se procedió a seleccionar y descargar manualmente los metadadatos de varios RAs (Juegos Serios) de diferentes tópicos (matemática, física, lenguaje, historia y geografía), para ser utilizados como datos de entrada.

Figura 4.1 Lista de RA de diferentes tópicos

En el desarrollo de las pruebas se toman en cuenta tres parámetros que cumplen una función específica dentro del sistema: el método de similitud usado para la comparación de cadenas, el umbral de semejanza entre el tema buscado y los RA, y el número de tramas usadas en la solución final. Se consideraron esos parámetros, ya que tienen una gran influencia en la construcción de los JSEs. Con respecto al resto de parámetros de ACO, se usó como referencia las investigaciones de [28], en las cuales existen recomendaciones sobre que parámetros utilizar para lograr un correcto desempeño del algoritmo ACO.

Método de similitud usado para comparar las cadenas: Durante el proceso de comparación de atributos se hace uso de un método que realiza la función de comparar la similitud entre cadenas de caracteres, con el fin de evaluar la cercanía de los atributos extraídos de un RA con el tema deseado. Particularmente se eligieron los métodos, "Levenshtein Distance" [29], "Jaro-Winkler Distance" [30] y "Jaccard Index" [31]. El

método de "Jaro-Winkler" fue elegido porque es uno de los métodos más utilizados que cae dentro de la categoría de los basados en el cálculo de la distancia para la comparación de cadenas. "Levenshtein" es un método clásico de comparación de cadenas que se basa en cuantas transformaciones hay que realizar en dos cadenas para que sean iguales, dicho número determina la distancia entre ellas. El método de "Jaccard" estudia el número de "tokens" que conforman cada cadena para realizar la comparación. Por otro lado, "Jaccard y Jaro-Winkler" arrojan como resultado un número en el rango "[0,1]", el cual determina la cercanía entre la cadenas comparadas. En el caso de "Levenshtein" se debió normalizar dicho número. Teniendo esto en mente, es necesario determinar cuál de los métodos de comparación de cadenas es el correcto para ser implementado en el sistema.

Umbral de semejanza: De los 15 atributos presentes en el tipo de temática que se busca, solo 4 son considerados de mayor importancia (título, lenguaje, descripción, y palabras clave), ya que es en estos atributos en donde se encuentra verdaderamente definido el tema del RA. Para esos atributos, durante el proceso de comparación se determina la similitud entre las cadenas de caracteres que conforman esos atributos en el RA con el tema buscado, generándose un número en el rango "[0,1]", donde 0 representa la inexistencia de similitud entre las cadenas y 1 que las cadenas son exactamente iguales. Teniendo este valor como referencia, es posible definir un umbral de semejanza entre los atributos comparados para determinar cuándo se consideran parecidos o no. Por ejemplo, si el promedio de la función para los cuatro atributos al compararse el tema buscado con el RA arroja un valor mayor a 0.6 y el umbral de semejanza es 0.6, entonces podemos concluir que hay similitud entre ellos, siendo este el índice de similitud del RA. Así, el umbral de semejanza debe ser analizado por lo sensible que es con la decisión sobre si un RA es similar al tema buscado o no.

Número de subtramas de la solución final: Se definió anteriormente que para que un juego sea considerado como solución final sin necesidad de pasar por ACO, es necesario que su índice de similitud sea mayor al umbral de semejanza (por ejemplo, 0.850). Teniendo esto en mente, al ser invocado el algoritmo ACO es porque el índice de similitud de cualquier RA no cuenta con un índice de similitud óptimo. Por esa razón, ACO construye una solución final compuesta por un número de subtramas, de tal manera de fusionar varios RAs que, en conjunto, generen un valor de similitud que cumpla con la temática buscada en un momento determinado (la complementación de sus atributos da un índice de similitud superior al umbral de semejanza). En ese sentido, la longitud máxima de subtramas permitida para los JSE debe ser estudiada.

A continuación se especifican tres escenarios de prueba (casos de uso) del sistema, cada escenario está diseñado para probar la sensibilidad del sistema a la hora de realizar cambios en los anteriores parámetros, evaluándose los resultados finales obtenidos con el algoritmo ACO, los cuales pueden ser:

Resultado óptimo: El juego generado por el algoritmo ACO cumple a cabalidad con la temática actual (el índice de similitud del JSE final es mayor al umbral de semejanza).

Resultado aceptable: El juego generado por el algoritmo ACO tiene algo de relación con la temática actual (el índice de similitud del JSE final es cercano al umbral de semejanza).

Resultado erróneo: El juego generado por el algoritmo ACO no tiene ninguna relación con la temática actual del curso (el índice de similitud del JSE es pequeño).

Además, para la evaluación de los escenarios se consideraron tres cursos. A saber:

Curso 1: se parte de la hipótesis que en el SaCI es necesario impartir una clase sobre el lenguaje y la comunicación verbal, por lo que se define un tema de aprendizaje que cuenta con la siguiente estructura: Materia: castellano, curso a aprender: literatura, tema: comunicación verbal. A partir de esta información, se genera una tabla 4.1 que contiene la información del curso, la cual será utilizada como punto de comparación con los metadatos obtenidos de los RAs.

Tabla 4.1 Datos del tema deseado en el Curso 1

LOM	**Tema Deseado**
Title	el lenguaje
Language	es
Description	comunicación verbal
Keyword	español, lenguaje, verbal, comunicación, habla
Coverage	universal
Format	html5, javascript
TypicalAgeRange	14
Difficulty	medium
Duration (minutes)	120
InteractivityLevel	medium
SemanticDensity	high
IntendedEndUserRole	tutor
Context	independent
CognitiveProcess	communicate
Cost	no

Curso 2: se parte de la hipótesis que en el SaCI es necesario impartir una clase sobre la raíz cuadrada, por lo que se define un tema de aprendizaje que cuenta con la siguiente estructura: Materia: matemáticas, curso a aprender: álgebra, tema o trama: raíz cuadrada. A partir de esta información, se genera la tabla 4.2 que contiene la información del curso, la cual será utilizada como punto de comparación con los metadatos obtenidos de los RA.

Tabla 4.2 Datos del tema deseado en el Curso 2

LOM	**Tema Deseado**
Title	la raíz cuadrada
Language	es
Description	raíz cuadrada
Keyword	raíz cuadrada, algebra, matemática
Coverage	universal
Format	html5, flash
TypicalAgeRange	12
Difficulty	easy
Duration (minutes)	120

InteractivityLevel	medium
SemanticDensity	low
IntendedEndUserRole	learner
Context	schoolmate
CognitiveProcess	understand
Cost	no

Curso 3: Se parte de la hipótesis que en el SaCI es necesario impartir una clase sobre fracciones, por lo que se define un tema de aprendizaje que cuenta con la siguiente estructura: Materia: matemáticas, curso a aprender: fracciones, tema o trama: fracciones equivalentes (ver Tabla 4.3).

Tabla 4.3 Datos del tema deseado en el Curso 3

LOM	**Tema Deseado**
Title	fracciones combinadas
Language	es
Description	emparejar fracciones
Keyword	fracciones, fracciones equivalentes, fracción, matemática, algebra
Coverage	universal
Format	html, jpeg
TypicalAgeRange	11
Difficulty	easy
Duration (minutes)	30
InteractivityLevel	high
SemanticDensity	high
IntendedEndUserRole	mixed
Context	schoolmate
CognitiveProcess	prove
Cost	no

4.2. Escenario Nº 1: Análisis de los Métodos de Comparación de Cadenas

El objetivo del primer escenario consiste en estudiar el funcionamiento de varios métodos de comparación de cadenas. Para llevar a cabo este escenario se eligieron los métodos de comparación de cadenas indicados anteriormente: "Levenshtein", "Jaro-Winkler" y "Jaccard". Con el fin de decidir cuál de ellos realiza mejor la tarea de comparar atributos a la hora de procesar RA.

Para la prueba con cada uno de los métodos de comparación de cadenas de caracteres, después de 30 corridas del algoritmo, y para el resto de parámetros de ACO con sus mejores valores, se generan los siguientes resultados promedios (ver Tabla 4.4).

Tabla 4.4 Resultados de las pruebas del Escenario Nº 1

		óptimos	aceptables	erróneos
Jaro-Winkler	Curso 1	100%	0%	0%
	Curso 2	43%	30%	27%
	Curso 3	87%	7%	6%
Jaccard	Curso 1	97%	0%	3%
	Curso 2	23%	30%	47%
	Curso 3	80%	7%	13%
Levenshtein	Curso 1	80%	17%	3%
	Curso 2	33%	37%	30%
	Curso 3	83%	10%	7%

Luego de finalizadas las pruebas podemos notar que el método basado en edición de distancias "Jaro-Winkler" presenta durante los tres casos, mejores resultados en comparación con los otros métodos. A primera instancia podríamos decir que un método basado en edición de distancias es el adecuado para ser implementado en el proceso de comparación. Por lo tanto se llegó a la conclusión de que "Jaro-Winkler" sería una elección que generaría resultados más precisos en pruebas posteriores. En general, el sistema requiere de un método de comparación de cadenas bastante preciso, debido a que el método forma parte del proceso de comparación de los atributos que tienen mayor peso sobre el índice de similitud de los RA procesados (título, lenguaje, descripción y palabras clave).

En general, lo que nos indican los diferentes métodos de comparación es su eficiencia para comparar las cadenas en los atributos de nuestros metadatos, siendo el mejor en nuestro contexto "Jaro-Winkler" ya que puesto a prueba en competencia con los otros métodos generó la mayor cantidad de resultados óptimos.

4.3. Escenario Nº 2: Umbral de Semejanza

El objetivo del segundo escenario consiste en evaluar la sensibilidad del umbral de semejanza.

Para esta prueba se definen varios valores de umbral: 0.4, 0.65, 0.8, y 0.95. Luego de 30 corridas del algoritmo para cada caso, se generan los siguientes resultados (ver Tabla 4.5).

Tabla 4.5 Resultados de las pruebas del Escenario Nº 2

		óptimos	aceptables	erróneos
0.4	Curso 1	0%	0%	100%
	Curso 2	0%	0%	100%
	Curso 3	0%	0%	100%
0.65	Curso 1	77%	6%	17%
	Curso 2	20%	33%	47%
	Curso 3	50%	23%	27%

0.8	Curso 1	100%	0%	0%
	Curso 2	40%	30%	30%
	Curso 3	87%	10%	3%
0.95	Curso 1	93%	0%	7%
	Curso 2	27%	20%	53%
	Curso 3	77%	13%	10%

Al finalizar con las pruebas, se puede notar que a medida que se realiza un aumento en el umbral de semejanza, se genera un aumento en los resultados óptimos y una disminución en los resultados erróneos. Particularmente, para valores menores a 0.4, el algoritmo es incapaz de obtener resultados óptimos o aceptables. Los resultados tienen sentido, debido a que al aumentar el umbral, el algoritmo se vuelve más estricto con respecto a lo que es considerado similar entre dos cadenas de caracteres. De tal manera que las tramas usadas para construir los Juegos Serios que se desvíen aunque sea un poco del tema buscado, son considerados erróneos. Así, el algoritmo ACO los va descartando naturalmente. Esta afirmación es cierta hasta cierto punto, ya que valores muy estrictos de umbral (0.95) tienen un impacto negativo con respecto a los resultados óptimos generados, ya que no permiten explorar con tramas que no sean tan semejantes (0.8) para construir Juegos Serios. Solo para el curso 2 se llegó a obtener peores resultados (menos óptimos y aceptables Juegos Serios) para el umbral 0.8, debido a lo específico del tema aunado al pequeño tamaño de la muestra de RAs que sirven de base de datos. Tomando en consideración lo anterior, se llegó a la conclusión de que para que el algoritmo genere JSEs cuya temática se acerque bastante al tema buscado, es necesario utilizar un umbral de semejanza cuyo valor esté cercano a 0.80.

4.4. Escenario Nº 3 Número de subtramas de la solución final

Para el último escenario, es necesario evaluar el número máximo de subtramas permitidas para generar un JSE con el algoritmo ACO. Es importante recordar que al finalizar el algoritmo, se realiza una fusión entre las subtramas escogidas y una integración de sus atributos. Si la puntuación de dicha integración iguala o supera el índice de similitud predefinido, concluimos que el Juego Serio generado cumple a cabalidad con el tema que se desea impartir. Para este escenario se toman los mejores valores de los otros parámetros para realizar la evaluación.

Para la prueba con diferente número máximo de subtramas permitida (2, 3, 4 y 5). Después de 30 corridas del algoritmo con cada número máximo de subtrama, y con el resto de parámetros de ACO con sus mejores valores, se generan los siguientes resultados promedios (ver Tabla 4.6).

Tabla 4.6 Resultados de las pruebas del Escenario N° 3

		óptimos	aceptables	erróneos
2	Curso 1	53%	40%	7%
	Curso 3	60%	30%	10%
3	Curso 1	57%	43%	0%
	Curso 3	27%	53%	20%
4	Curso 1	73%	10%	17%
	Curso 3	47%	20%	33%
5	Curso 1	43%	0%	57%
	Curso 3	17%	3%	80%

Primero que nada es importante señalar que debido a la base de datos proporcionada, no se realizaron experimentos relacionados con el curso 2. La razón de esto tiene que ver con lo específico del tópico y al tamaño de la muestra de RAs utilizada.

La correcta integración de dos o más subtramas se encuentra directamente relacionada con el tamaño de la muestra, mientras más grande la muestra de RAs procesada por el algoritmo, las posibilidades de generar una fusión óptima de subtramas aumenta. Al analizar los datos de la tabla se puede notar que a medida que aumenta el número máximo de subtramas, también aumenta la cantidad de resultados erróneos, esto se debe a que en una muestra pequeña de RAs, mientras aumenta el número de subtramas, la cantidad de Juegos Serios que cumplan con una temática disminuye, de tal manera que el algoritmo se ve en la obligación de llenar esos espacios vacíos con Juegos Serios que no cumplan con el tema buscado. Ahora bien, el mejor JSE conseguido al aumentar el número máximo de subtramas (0.92) es mejor que con un número de subtramas pequeño (0.89), lo que nos indica que a pesar de porcentualmente conseguir pocos buenos JSE, los que se consiguen son mejores porque se tienen más subtramas para construir JSE más precisos. Otro aspecto importante a resaltar es que al aumentar el número máximo de subtramas el tiempo de ejecución del algoritmo ACO aumenta, ya que el espacio de posibles soluciones que debe explorar para construir los posibles JSEs es más grande (más combinaciones posibles de subtramas).

Los tres escenarios permitieron analizar los parámetros adecuados a usar en el STE. La combinación "optima" de parámetros depende del objetivo que se quiera lograr en el salón de clases en un momento determinado. Si se quiere ser estricto en cuanto a la temática a buscar, entonces se requieren un buen número de tramas y umbrales altos, con un método de comparación de cadenas exigente. Por el contrario, si se quiere ser más permisible en cuanto a que tan exigente se quiere ser con el Juego Serio y su relación a la temática, pero mejorando los tiempos de búsqueda, entonces se puede usar métodos de comparación de cadenas menos exigentes, con umbrales de similitud bajos y numero de subtramas bajo.

Un ejemplo de resultado de JSE óptimo dado por el algoritmo ACO es mostrado en la Figura 4.2. Ese JSE generado cumple con el tema actual de la clase de fracciones, ya que la puntuación de la fusión de subtramas supera el umbral de semejanza predefinido (ver Figura 4.2).

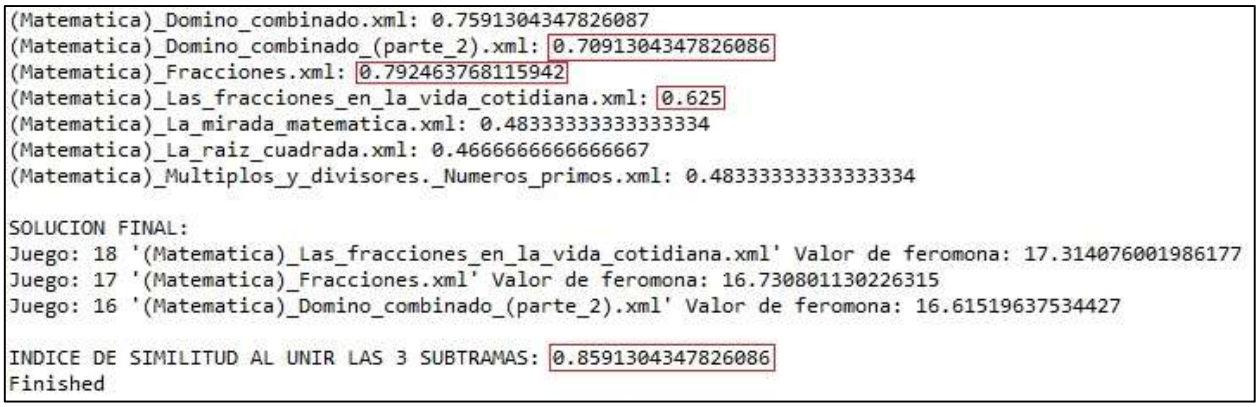

Figura 4.2 Ejemplo de resultado óptimo para el Escenario N° 3

En esa figura se puede ver que se propone un JSE compuesto por tres tramas (juego 18, 17 y 16), que al combinar sus atributos da una semejanza de 0,85.

Capítulo 5

Comparación con otros Trabajos

Durante el siguiente capítulo se llevara a cabo una comparación de cualidades entre el proyecto desarrollado y una serie de trabajos cuyo contenido, de alguna manera, tiene relación con un motor de videojuegos que permite la emergencia de Juegos Serios. Para la realización de dicha comparación se toman en cuenta los siguientes criterios:

- ¿Es un Juego Serio? Con esta pregunta se quiere determinar si la propuesta se enfoca principalmente en el uso de Juegos Serios, juegos orientados principalmente a la enseñanza.
- ¿Tiene o confiere capacidades adaptativas al juego? Con esta pregunta se quiere determinar si se proponen capacidades adaptativas al juego, con el objetivo de mejorar la experiencia del usuario.
- ¿Permite algún tipo de Emergencia? Con esta pregunta se quiere determinar si las propiedades del juego se van desplegando de manera espontánea, autónoma y sin leyes explicitas, adecuándose a los jugadores.
- ¿Forma parte de un Motor de Juegos? Con esta pregunta se quiere determinar si la propuesta forma parte del núcleo general de un Motor de Juegos, siendo uno de sus componentes.

La siguiente tabla compara nuestra propuesta con otros trabajos, desde el punto de vista de esos criterios anteriormente nombrados.

Tabla 5.1 Comparación con otros Trabajos

Criterio	[19]	[25]	[32]	[33]	[34]	[35]	[36]	Presente Trabajo
¿Es un Juego Serio?			X		X	X		X
¿Tiene o confiere capacidades adaptativas al juego?	X	X		X		X	X	X
¿Permite algún tipo de emergencia?				X			X	X
¿Forma parte de un Motor de Juegos?						X		X

En [19] se propone un juego de estrategia que consiste en agentes que basan su comportamiento en el de forrajeo y la forma defensiva de las colonias de abejas, para adaptarse a un entorno humano. De esta forma, el juego consta de múltiples agentes cooperativos autónomos. La capacidad de adaptación fue medida a través de un experimento empírico sobre un entorno simulado, en donde se realizaron una serie de evaluaciones sobre el comportamiento de los agentes, teniendo en cuenta las decisiones tomadas (correctas e incorrectas) y el tiempo de decisión, en contraste con un conjunto de parámetros óptimos ya establecidos. A su vez, la calidad de los agentes fue medida a través de una encuesta realizada a un grupo de expertos en juegos de estrategia luego de la interacción con el sistema.

En [25] se utiliza ACO para explorar el entorno y comunicar información sobre los recursos, con el fin de proporcionar capacidades adaptativas al juego. Se realizó un estudio empírico sobre un entorno simulado para evaluar el sistema en diferentes niveles de dificultad.

En [32] se propone un proceso cuyo objetivo es garantizar la correcta implementación de los aspectos pedagógicos (actividades de aprendizaje y contenido, aprendizaje esperado, etc.) en la producción de Juegos Serios. Una vez aplicado el proceso de producción, midieron la calidad del prototipo aplicando los Juegos Serios en un salón de clases. Luego de un periodo académico, evaluaron la evolución de los estudiantes que interactuaron con los juegos, en comparación con los que siguieron un proceso educativo tradicional.

En [33] se describen dos juegos que demuestran la viabilidad del uso de algoritmos heurísticos para problemas de optimización combinatoria: el primero es un juego de estrategia, que hace emerger recursos en tiempo real usando ACO. El segundo es un juego de carreras, donde la inteligencia artificial adapta características numéricas como la velocidad, la agilidad y el poder de salto, para mejorar el desempeño del corredor durante la competencia con otros jugadores.

[34] propone una arquitectura para el desarrollo de videojuegos de tipo educativo, para alentar a los usuarios inexpertos a interactuar con algoritmos de inteligencia artificial. El juego le permite al usuario experimentar a través de cambios en los parámetros de los algoritmos de inteligencia artificial, entre los cuales se encuentra ACO.

En [35] proponen el desarrollo de Juegos Serios usando un motor adaptable, el cual usa una técnica de aprendizaje automático para el proceso de adaptación del Juego Serio a diferentes jugadores. El motor adaptativo consta de reglas establecidas basadas en la descripción de las competencias del jugador, su estilo de aprendizaje, estado cognitivo, entre otras cosas.

En [36] se optimizan rutas con Beam-ACO y Ant-Q, para hacer emerger de forma dinámica una ruta personalizada, basada en datos de juegos anteriores, que sirva como apoyo a los usuarios a la hora de cumplir con los objetivos del juego.

La principal diferencia de nuestra propuesta y los trabajos anteriores es que nuestra propuesta forma parte de un MJSE. Además, es el único que permite hacer emerger tramas/secuencias en un JSE, según las características del contexto donde es usado. Para ello, usa un algoritmo ACO que le permite generar posibles alternativas de JSEs, y seleccionar aquellas que cumplan mejor con los atributos que caracterizan al tema deseado.

En general, los trabajos anteriores confieren capacidades adaptativas, pero no están orientados a formar parte de un MJSE. Por otro lado, algunos permiten emerger comportamientos en la dinámica del juego

derivado de su uso, realizando principalmente emergencias de estrategias y de propiedades, y no permiten la emergencia de secuencias en los juegos guiada por el contexto, como en nuestro caso. Ahora bien, no hay ninguno orientado a JSE, y de los pocos orientados a Juegos Serios, no tienen en su mayoría capacidades adaptativas ni son pensados para formar parte de MJS.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

El uso de JSEs que se adecuen a un contexto dado (por ejemplo, a un SaCI), ayudan al proceso que esté ocurriendo en un momento determinado en ese contexto (por ejemplo, al proceso educativo de SaCI), haciéndolo más emocionante, entretenido y dinámico, motivando así a los actores en ese ambiente. Para ello, se requiere de un MJSE que permita las diferentes emergencias. Ningún trabajo previo ha desarrollado un Submotor de Trama Emergente, y menos en el contexto de un SaCI.

El objetivo de nuestro trabajo consiste en implementar un subsistema que se encargue de procesar una serie de Juegos Serios, para luego hacer emerger dinámicamente un JSE óptimo para un tema y momento determinado. Para ello se ha usado un algoritmo ACO, que permite la emergencia de tramas en un JSE a partir del manejo ordenado y sistemático de un conjunto de subtramas recuperadas de repositorios de RA, vinculadas a un contexto y dominio deseado. El algoritmo ACO, de manera autónoma, selecciona las mejores subtramas que serán usadas en el JSE.

El diseño, desarrollo e implementación del *módulo de recuperación de trazas*, encargado del procesamiento de los RA para definir qué tan parecido es el contenido de los Juegos Serios con respecto a la temática que se esté buscando, y del *módulo de emergencia de secuencias* basado en ACO, encargado de construir un JSE que cumpla con un objetivo específico (temática deseada); mostraron ser una solución factible para la elección de RA a partir de una temática predefinida.

Específicamente, nuestra herramienta permite extraer la información contenida en los metadatos de un RA, y a partir de dicha información, asignar una puntuación según su similitud con respecto a un tópico determinado. Al hacer uso de esta información, y con la ayuda de ACO, ocurre un proceso de emergencia de JSEs que tienen relación con la temática actual de un SaCI. La fusión de subtramas hace posible generar Juegos Serios que cumplen a cabalidad con el tema desarrollado en el salón de clases.

El desarrollo de los experimentos permitió determinar la sensibilidad del STE al método de comparación de cadenas, al número de subtramas usadas y al umbral de semejanza. Dichos valores repercuten altamente en la calidad de los JSE obtenidos. En ese sentido, la elección de los valores de dichos parámetros depende del nivel de exigencia que se quiera con respecto al JSE a usar en un salón de clases, en cuanto a su semejanza o no con la temática tratada en el salón, y al tiempo que se le dará al algoritmo ACO para proveerlo. En cuanto a los trabajos anteriores, no se consiguieron en la literatura propuestas que permitan la emergencia de tramas en un JSE para adecuarlo a una temática de un curso en un salón de clases, y que además, formen parte de un MJSE.

6.2. Trabajo Futuro

Antes que nada, se debe probar el prototipo desarrollado de ACO en diferentes contextos, en particular en un SaCI real, para evaluar su impacto sobre los actores en ese ambiente. Para ello, se deben usar métricas que avalúen el uso de los JSEs producidos en el proceso de enseñanza, y particularmente su impacto en dicho proceso. También, es necesario analizar la integración del algoritmo con otros tipos de emergencias posibles en un JSE (comportamiento, propiedades, entre otros).

Otro aspecto importante por realizar con nuestra herramienta es evaluar la sensibilidad e influencia de los parámetros que forman parte de las funciones básicas del macroalgoritmo de ACO. Dichos parámetros forman parte del proceso realizado por las hormigas artificiales, y tiene que ver con los procesos de inicialización del algoritmo, actualización de feromonas, función de transición, etc. Para nuestras pruebas se tomaron los valores de parámetros de ACO aconsejados en [18], ya que nuestro interés fue analizar en detalle los parámetros que afectaban directamente la concepción de JSE con el algoritmo ACO (umbral de semejanza, método de comparación de cadenas y número máximo de tramas permitido para la construcción de JSEs).

Bibliografía

[1] Revuelta, T. (2015). Desarrollo y aplicación del algoritmo de optimización basado en colonia de hormigas (aco) para la resolución del problema del viajante asimétrico (atsp), Grado en ingeniería de organización industrial, Universidad De Valladolid, Escuela de Ingenierías Industriales, España.

[2] Altamiranda, J. (2012). Reconocimiento de Patrones Adaptativos en Proteínas Amiloideas Usando Expresiones Regulares, Doctorado en ciencias aplicadas, Proyecto de grado doctoral, Universidad de los Andes, Facultad De Ingeniería, Mérida, Venezuela.

[3] Altamiranda, J. (2010). Algoritmos de optimización basados en colonias de hormigas aplicados al problema de asignación cuadrática y otros problemas relacionados, Trabajo final para alcanzar el grado de licenciado en ciencias de la computación, Universidad Nacional de San Luis, Facultad de Ciencias Físico Matemáticas y Naturales, San Luis, Argentina.

[4] Petridis, P., Duenwell, I., Panzoli, D., Arnab, S., Aristidis, P., Hendrix, M. and Freitas, S. (2012). Game Engines Selection Framework for High-Fidelity Serious Aplications, Journal of Interactive Worlds, Vol. 2012, Article ID 418638.

[5] Fuentes, I. (2012). Psicología y realidad virtual, Videojuegos terapéuticos, Revista Muy Interesante, vol. 12, pp. 102–105, México.

[6] Gracia, B., Sanagustín, G., and Romero, S. (2015). Análisis de motores gráficos y su aplicación en la industria, Technical report, TecsMedia, División de Tecnologías Multimedia, Instituto Tecnológico de Aragón (ITAINNOVA), España.

[7] Agustín, H., and Garde, M. (2016). Desarrollo de un motor de eventos para videojuegos, Technical report, Red de Interconexión de los Recursos Informáticos, España.

[8] Sweetser, P. (2006). An Emergent Approach to Game Design – Development and Play, A thesis submitted for the degree of Doctor of Philosophy, School of Information Technology and Electrical Engineering, The University of Queensland, Australia.

[9] Aguilar, J., Cardozo, J., González, C. and Rengifo, B. (2013). Una aproximación a los Juegos Emergentes, Metrópolis, Simulador de Ciudades Autogestionadas, 47va Conferencia Latinoamericana de Informática.

[10] Aguilar, J., Altamiranda, J., and Chávez, D. (2016). Extensiones a Metrópolis para una Emergencia Fuerte, Revista Venezolana de Computación, vol. 3, no. 2, pp. 38-46.

[11] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics Part B, vol. 26, pp. 29–41.

[12] Aguilar, J. (2014) Introducción a los sistemas emergentes, Technical report, Universidad de los Andes, Mérida, Venezuela.

[13] Cruz-Lara, S., Fernández, B., and Vaz de Carvalho, C. (2013). Enfoques innovadores en juegos serios, IEEE VAEP RITA, vol. 1, pp. 19–21, España.

[14] Salvat, B. (2009). Certezas e interrogantes acerca del uso de los videojuegos para el aprendizaje, Communication, vol. 7, pp. 251–264, España.

[15] KissFlow: Rapid Application Development: Changing How Developers Work. (2018). [Website]. Recuperado de: https://kissflow.com/rad/rapid-application-development/

[16] LucidChart: 4 Phases of Rapid Application Development Methodology. (2018). [Website]. Recuperado de: https://www.lucidchart.com/blog/rapid-application-development-methodology

[17] Bellotti, F., Berta, R. and De Gloria, A. (2010). Designing Effective Serious Games: Opportunities and Challenges for Research, Special Issue: Creative Learning with Serious Games, Intl. Journal of Emerging Technologies in Learning (IJET), Vol. 5, pp. 22-35.

[18] Chan, R., Zhang, H., Tao, X. (2017). Serious Game Design for Stroke Rehabilitation, Intl Journal of Information Technology, Vol. 23.

[19] Daylamani-Zad, D., Graham, L., and Paraskevopoulos, I. (2018). "Chain of command in autonomous cooperative agents for battles in real-time strategy games", Journal of Computers in Education, Vol. 6, pp 1–32.

[20] Millqvist, L., Brusk, J., and Björn Berg, M. (2018). "Ludonarrativ Dissonans Påverkan På Ett Spels Trovärdighet," Examensarbete inom huvudområdet Informationsteknologi, Vårtermin.

[21] Karlsson, P., Gunnarsson, G., and Kristensen, L. (2018). "Spelares Engagemang för Narrativet I Plattformsspel," Examensarbete inom huvudområdet Medier, estetik och berättande Vårtermin.

[22] Vallejo, D., Martín, C. (2013). Desarrollo de Videojuegos: Arquitectura del Motor de Videojuegos (2ª Ed.). España: Universidad Castilla la Mancha.

[23] Aguilar, J., Altamiranda, J., Díaz, F., and Mosquera, D. (2016). "Motor de Juego Serios en ARMAGAeco-c," Revista UNET, Vol. 28, pp. 100-110.

[24] Aguilar, J., Altamiranda, J., and Díaz, F. (2018). "Design of a Serious Emerging Games Engine Based on the optimization Algorithm of Ant Colony," DYNA, Vol. 85, no 206, pp. 311-320.

[25] Chen, X., Ong, Y., Feng, L., Lim, M., Chen, C., and Ho, C. (2013). "Towards believable resource gathering behaviours in real-time strategy games with a memetic ant colony system, Procedia Computer Science," vol. 24, pp. 143–151, 2013.

[26] Subbaraj, S., and Savarimuthu, P. (2014). "EigenTrust-based non-cooperative game model assisting ACO look-ahead secure routing against selfishness," Proceedings EURASIP Journal on Wireless Communications and Networking, pp. 1-20.

[27] IEEE LTSC LOM: Draft Standard for Learning Object Metadata. (2010). [Website]. Recuperado de: https://ieeexplore.ieee.org/document/5445243

[28] Dorigo, M., Stützle, T. (2004). Ant Colony Optimization. Cambridge, Massachusetts: The MIT Press.

[29] Schulz, K., Mihov, S. (2002). "Fast String Correction with Levenshtein-Automata". International Journal of Document Analysis and Recognition. 5 (1): 67–85.

[30] AppaloosaStore: String Similarity Algorithms Compared. (2018). [Website]. Recuperado de: https://medium.com/@appaloosastore/string-similarity-algorithms-compared-3f7b4d12f0ff

[31] Mayank, M.: String Similarity – the basic know your algorithm guide. (2019). [Website]. Recuperado de: https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227

[32] Barajas, A., Álvarez, F., Muñoz, J., and Oviedo, A. (2016). "Process for modeling competencies for developing serious games," Revista Electrónica de Investigación Educativa, vol. 18, no. 3, pp. 146-160.

[33] Jamieson, P., Grace, J., Hall, J., and Wibowo, A. (2013). "Metaheuristic Entry Points for Harnessing Human Computation in Mainstream Games," Proceedings of International Conference on Online Communities and Social Computing (OCSC 2013), vol. 8029, pp. 156–163.

[34] Martínez, J., López, A., and Maldonado, M. (2015). "On the Use of Ant Colony Optimization for Video Games," Proceedings of Advances in Artificial Intelligence and Soft Computing. MICAI, vol. 9413, pp 238247.

[35] Rasim, T., Langi, A., Munir, S., and Rosmansyah, Y. (2016). "A survey on adaptive engine technology for serious games, Proceedings of International Seminar on Mathematics," Science, and Computer Science Education (MSCEIS 2015) AIP Conf. Proc. 1708, vol. 1708, no. 1, pp. 50003-1–50003-9.

[36] Tregel, T., Müller, P., Göbel, S., and Steinmetz, R. (2018). "Where's Pikachu: Route Optimization in Location-Based Games," Proceedings 10th International Conference on Virtual Worlds and Games for Serious Applications (VSGames).

Anexos

Anexo A. Repositorio del Proyecto.

En el siguiente enlace "https://bitbucket.org/andtreso/aco-seg-java/src/master" se encuentra alojado el repositorio del proyecto: **ACO Implementation for a Serious Emerging Games Engine** en el sitio web: bitbucket.org. Para próximas colaboraciones (ver Figura 0.1).

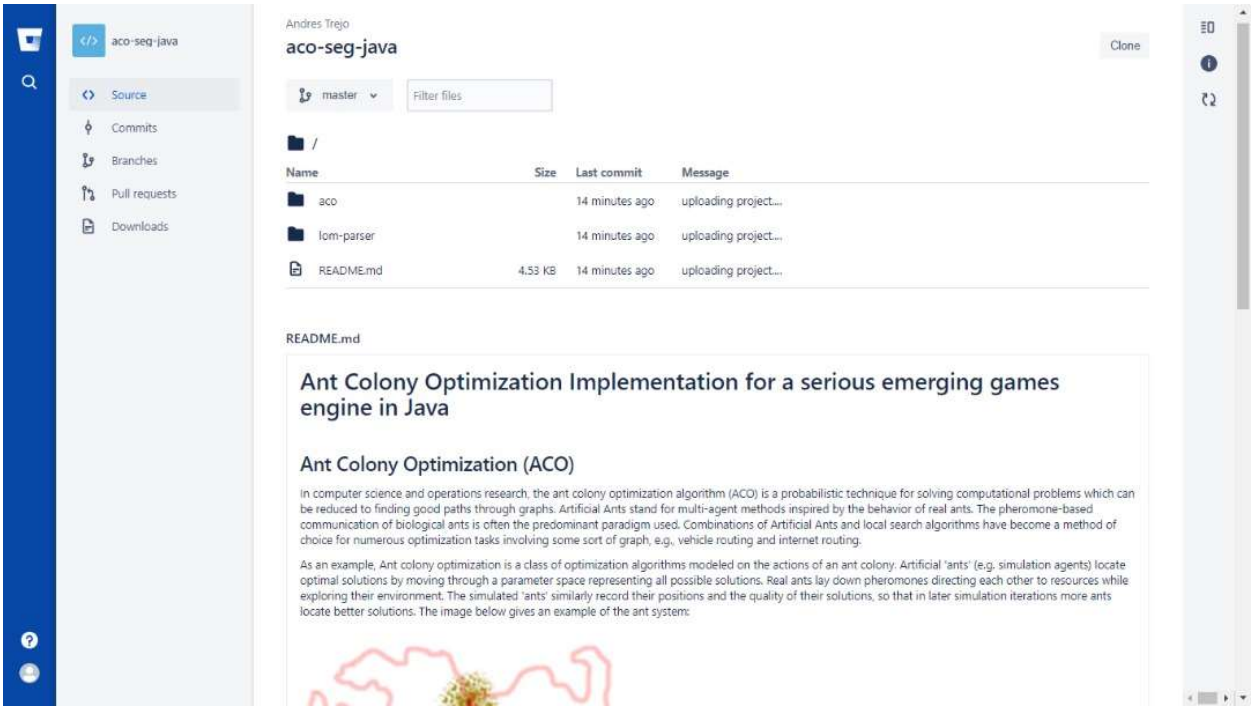

Figura 0.1 Repositorio del Proyecto

Anexo B. Funciones del Submotor de Trama Emergente.

A continuación se presenta la información de los métodos implementados en el Submotor de Trama Emergente, donde se exponen:

- Descripción: Información de que hace el método.
- Función: Se refiere al nombre asignado a la función para acceder al método.
- Prueba: Es el archivo en donde se encuentra un ejemplo del método.
- Ejemplo: Es una prueba corta de cómo puede ser utilizada la función.
- Parámetros: Son todos los parámetros que influyen en la función.
- Retorna: Son todos los parámetros que retorna la función.

Descripción	**Constructor de la clase Environment, accede a sus métodos internos.**
Función	Environment(Graph);
Prueba	Environment.java
Ejemplo	Environment e = new Environment(graph);
Parámetros	• graph es una clase auxiliar que contiene toda la información del grafo a procesar.
Retorna	No retorna nada.

Descripción	**Función que genera una matriz nxn de "vecinos cercanos". Asocia cada nodo del grafo a una lista de nodos ordenados por cercanía de acuerdo al peso de los arcos que los conectan.**
Función	void generateNearestNeighborList();
Prueba	Environment.java
Ejemplo	Environment e; e.generateNearestNeighborList();
Parámetros	No tiene parámetros.
Retorna	No retorna nada. La matriz queda guardada en una variable del entorno.

Descripción	**Crea una población de k hormigas utilizadas para buscar soluciones en el entorno.**
Función	void generateAntPopulation();
Prueba	Environment.java
Ejemplo	Environment e; e.generateAntPopulation();
Parámetros	No tiene parámetros.
Retorna	No retorna nada. La lista de hormigas queda guardada en una variable del entorno.

Descripción	**Función encargada de crear e inicializar dos matrices:** • La matriz que contiene la información de feromonas de cada arco del grafo. • La matriz de probabilidades utilizada por las hormigas para realizar la transición de un nodo a otro.
Función	void generateEnvironment();
Prueba	Environment.java
Ejemplo	Environment e; e.generateEnvironment();
Parámetros	No tiene parámetros.
Retorna	No retorna nada.

Descripción	**Coloca a cada hormiga a construir una solución en el entorno.**
Función	void constructSolutions();
Prueba	Environment.java
Ejemplo	Environment e; e.constructSolutions();
Parámetros	No tiene parámetros.
Retorna	No retorna nada.

Descripción	**Actualiza el nivel de feromonas del grafo en dos pasos:** • Se evapora la cantidad de feromonas de los arcos y nodos del grafo de acuerdo a un parámetro predefinido. • Cada hormiga del entorno deposita una cantidad de feromonas en los arcos y nodos del grafo presentes en su recorrido actual.
Función	void updatePheromone();
Prueba	Environment.java
Ejemplo	Environment e; e.updatePheromone();
Parámetros	No tiene parámetros.
Retorna	No retorna nada.

Descripción	**Una vez que la colonia concluye su trabajo, se hace un recorrido a través de todos los nodos del grafo, seleccionándose los nodos con mayor nivel de feromona. Luego se realiza una fusión del contenido de dichos nodos para generar una solución final.**
Función	void calculateStatistics(int, String[]);
Prueba	Environment.java
Ejemplo	Environment e; e.calculateStatistics(traces, topic);
Parámetros	• traces es el número de subtramas requerido para generar la solución final.

	• topic es un arreglo que contiene la información de 15 atributos que representan la temática que se busca.
Retorna	No retorna nada, se muestran los resultados del algoritmo.

Descripción	**Constructor de la clase Ant, accede a sus métodos internos.**
Función	Ant(int, Environment);
Prueba	Ant.java
Ejemplo	Ant a = new Ant(tourSize, Environment);
Parámetros	• tourSize es la cantidad de nodos del grafo a explorar. • Environment es el entorno en el cual la hormiga va a buscar soluciones.
Retorna	No retorna nada.

Descripción	**Resetea la lista de nodos visitados de una hormiga.**
Función	void clearVisited();
Prueba	Ant.java
Ejemplo	Ant a; a.clearVisited();
Parámetros	No tiene parámetros
Retorna	No retorna nada

Descripción	**Coloca a una hormiga en un nodo aleatorio del grafo y lo marca como visitado.**
Función	void startAtRandomPosition(int);
Prueba	Ant.java
Ejemplo	Ant a; a.startAtRandomPosition(phase);
Parámetros	• phase es el entero que representa la primera posición de la lista de nodos visitados de la hormiga. Normalmente este número siempre será 0.
Retorna	No retorna nada.

Descripción	**Función de transición utilizada por las hormigas para elegir que nodo visitar que no haya previamente visitado.**
Función	void goToNNListAsDecisionRule(int);
Prueba	Ant.java
Ejemplo	Ant a; a.goToNNListAsDecisionRule(phase);
Parámetros	• phase es el entero que representa la posición actual de la lista de nodos visitados de la hormiga.
Retorna	No retorna nada.

Descripción	**Finaliza el recorrido de la hormiga actualizando el valor de la longitud y el promedio de los índices de similitud de los nodos visitados en el recorrido.**
Función	void finishTour(int);
Prueba	Ant.java
Ejemplo	Ant a; a.finishTour(phase);
Parámetros	• phase es el entero que representa la posición actual de la lista de nodos visitados de la hormiga.
Retorna	No retorna nada.

Descripción	**Constructor de la clase Aco, accede a sus métodos internos.**
Función	Aco();
Prueba	Aco.java
Ejemplo	Aco a = new Aco();
Parámetros	No tiene parámetros.
Retorna	No retorna nada.

Descripción	**Función que aplica el algoritmo colonia de hormigas a una serie de recursos de aprendizaje.**
Función	void startAlgorithm(String, String[], int);
Prueba	Aco.java
Ejemplo	Aco a; a.startAlgorithm(filesPath, topic, traces);
Parámetros	• filesPath es la dirección de la carpeta en donde se encuentran los recursos de aprendizaje a procesar. • topic es el tópico que será utilizado para la comparación entre los recursos de aprendizaje y el tema deseado. • traces es el número de subtramas requerido para generar la solución final.
Retorna	No retorna nada.

Descripción	**Crea un grafo de recursos de aprendizaje para ser utilizado por el ACO.**
Función	void parseFiles(String, String[]);
Prueba	LomParser.java
Ejemplo	LomParser.parseFiles(myPath, topic);
Parámetros	• myPath es la dirección local de la carpeta donde se encuentran contenidos los recursos de aprendizaje a procesar. • topic es el tópico que será utilizado para la comparación entre el recurso de aprendizaje y el tema deseado.
Retorna	No retorna nada. Crea un grafo.

Descripción	**Realiza el parseo de un archivo xml.**
Función	String[] parseLOM(String);
Prueba	LomParser.java
Ejemplo	LomParser.parseLOM(file);
Parámetros	• file es el nombre del archivo .xml que contiene la información del recurso de aprendizaje.
Retorna	Retorna un arreglo con la información de 15 atributos que representan la temática del recurso de aprendizaje.

Descripción	**Compara los atributos de un recurso de aprendizaje con un tópico predefinido.**
Función	double compare(String[], String[]);
Prueba	ScoreModule.java
Ejemplo	ScoreModule.compare(topic, resource);
Parámetros	• topic es el tópico que será utilizado para la comparación entre el recurso de aprendizaje y el tema deseado. • resource es la información del recurso de aprendizaje.
Retorna	Retorna un número que representa el índice de similitud entre el recurso de aprendizaje y el tópico deseado.