UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS.

FACULTAD 4.

Título: "Agente resolvente de problemas de búsqueda heurística"

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Xiunellys Huerta Quintanilla

Tutores: Msc. Yuniesky Coca Bergolla

Ing. Juan Gabriel Valdés Díaz

Ing. Leduan Bárbaro Rosell Acosta

La Habana, Cuba 2017

"Año 59 de la Revolución"

“Si puedes soñarlo, puedes hacerlo”.

Walt Disney.

Declaración de autoría

Declaro ser la única autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Xiunellys Huerta Quintanilla

Autor

Msc. Yuniesky Coca Bergolla.

Tutor

Ing. Juan Gabriel Valdés Díaz

Tutor

Ing. Leduan Bárbaro Rosell Acosta

Tutor

Datos de contacto

Tutor: MSc. Yuniesky Coca Bergolla.

Email: ycoca@uci.cu

Graduado de Licenciatura en Ciencias de la Computación en la Universidad Central de Las Villas (UCLV) 2003. Defendió su maestría en Informática Aplicada en la Universidad de las Ciencias Informáticas (UCI) en 2007. Ha tutorado más de 15 tesis de grado relacionadas con temas de Inteligencia Artificial (IA). Actualmente se desempeña como Jefe de disciplina de IA en la UCI. Además, es árbitro de la Revista Cubana de las Ciencias Informáticas (RCCI).

Tutor: Ing. Juan Gabriel Valdés Díaz

Email: jgvaldes@uci.cu

Graduado de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas (UCI) 2016. Actualmente se desempeña como Jefe de la Línea de Desarrollo Web en el Centro VERTEX de la UCI.

Tutor: Ing. Leduan Bárbaro Rosell Acosta

Email: lbrosell@uci.cu

Graduado de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas (UCI) 2016. Actualmente imparte clases de Inteligencia Artificial 2 en la Facultad 4 de la UCI.

Dedicatoria

Mi tesis va dedicada a Maritza Quintanilla Izquierdo. Quien es la persona más importante en mi mundo, por la cual vivo y a la vez muero. Mami por tu forma de ser conmigo eres lo que más quiero. Eres mi timón, mi vela, mi mar, mi barca y mi remo.

Te amo siempre mi Mamma Bella.!!

Agradecimientos

Deseo agradecer primero que nada a la vida, al destino y a dios, por estar en este momento cumpliendo uno de los objetivos en mi vida, de ser profesional, y por darme la oportunidad de conocer personas tan lindas a lo largo de esta carrera.

A mi madre preciosa y perfecta que me ama y me acepta tal como soy, que siempre ha estado conmigo en todo, aceptando inclusos las estupideces que se me ocurren, ella que siempre me ha dado un sí como respuesta, que lucha por complacerme en todo y porque ella no come sin saber que yo lo he hecho.

A mis hermanitas divinas Xayu y Nany, que me adoran y sé que yo soy un ejemplo para ellas, pero recuerden hacer lo que yo digo no lo que yo hago.

A mi Tía Ilsis Elena, que como ella misma dice, por darme me ha dado hasta el nombre, sin duda alguna digo que no me alcanzará la vida para agradecer esta bella mujer que me ha dado la posibilidad de tener dos madres y ser ella mi padre. A Tío Oscar que lo adoro con la vida y me ha acogido como una más de su familia.

A mis primos Adita y O. Junior, que los quiero mucho y los he visto crecer y ser los desastritos que hoy son.

A mi abuelita Nena, mi abuelo Quintanilla, a mi Tía Mayra que la quiero mucho tal como es. Mi Tía Magaly y mis primillas Liennys y Lisney.

A Coki, como cariñosamente le digo, por realmente darme un apoyo sobrenatural y hacerse cargo de mi persona. Realmente profe, escogerlo a Ud como tutor ha sido una de mis mejores decisiones. ¡Gracias!

A mis co-tutores Juan Gabriel y Leduan B. Juan que desde el 20 de septiembre del 2012 hemos sido más que amigos, él ha sido uno de los mejores regalos que me llevo de esta universidad, un amigo como él realmente es una mina de oro. Gracias Juancillo veo. Leduan, quien le agradezco eternamente todo su esfuerzo y dedicación. Sabes que sin ti jamás lo hubiese logrado. Gracias por estar en el TEAM_XIU, ha sido genial contar contigo, realmente gracias por estar ahi siempre que te llame. ¡Los quiero muchote mis niños!

¡A todos todos todos mis amigos UCI! Los que conozco hace años, lo que conozco hace días, hace hora, a los que llamo y me cuelgan, a los que me dejan quema, los que no

me llaman nunca. A todos los que ya no son amigos, a todos los que me odian, los que están al tanto de mi vida. A todos gracias por estar conmigo en las buenas y mala, por los momentos de risa, de llanto y de fiesta. ¡Muchas Gracias!

Y por último y no menos importante a las personas que ya no se encuentran entre nosotros, pero aun así sé que me cuidan y velan por mí. ¡Los amo y los extraño!

Resumen

Las nuevas tecnologías de la información y las comunicaciones han incidido cada vez más en el desarrollo de sistemas inteligentes utilizados en la educación. Sin embargo, aún quedan lagunas en la utilización de los mismos en varias ramas de interés, un ejemplo de ello es la disciplina Inteligencia Artificial. La presente investigación se realiza a partir de la necesidad de contar con una herramienta informática de código abierto capaz de permitir el estudio práctico de algoritmos clásicos para la búsqueda de caminos mínimos y búsquedas locales. Estos algoritmos son analizados en el tema Métodos de Solución de Problemas, perteneciente a la asignatura Inteligencia Artificial 1 que se imparte en la Universidad de las Ciencias Informáticas. Para el desarrollo del software que se propone como solución, se utilizó el lenguaje Java, el *IDE NetBeans* y el *framework JavaFX* para la realización de la interfaz visual. En tanto, la metodología XP fue la utilizada para guiar el proceso ingenieril de la presente investigación. El principal aporte del trabajo es la construcción del software siguiendo el paradigma de desarrollo de agentes y estar diseñado para ser utilizado, además de su interfaz, para la reutilización de su código por parte de los estudiantes.

Palabras clave: agente; A*; búsqueda heurística; código abierto; escalador de colinas; inteligencia artificial.

Índice

Índice de imágenes

Índice de tablas

Introducción

El creciente desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC´s) ha cambiado la forma de vida de la humanidad, se evidencia su impacto en muchas áreas del conocimiento tales como la salud, telecomunicaciones, finanzas, política, administración, empresas, educación, vida cotidiana y diversión. En el área educativa, las TIC´s han demostrado que pueden ser de gran apoyo tanto para los docentes, como para los estudiantes. La implementación de la tecnología en la educación debe verse como una herramienta de apoyo y en ningún caso debe sustituir al maestro. Por lo general pretende ayudar al estudiante a tener más elementos visuales y auditivos, con el objetivo de enriquecer el proceso de enseñanza y aprendizaje. Sin embargo, en carreras afines a la informática, la tecnología debe aportar vías para la práctica, buscando desarrollar habilidades relacionadas con la programación y la resolución de problemas computacionales.

Estar a la vanguardia en los métodos de enseñanza y ofrecer oportunidades de prácticas innovadoras que apoyen la preparación de los estudiantes para su futura vida laboral, resulta ser una de las preocupaciones constantes de las universidades y centros de estudios superiores. De manera conjunta se ve afectada la forma de enseñar y aprender en estos recintos, debido al continuo desarrollo tecnológico que las organizaciones viven, particularmente en las áreas de almacenamiento de información, recuperación y comunicación. La enseñanza dentro de las universidades debe permanecer también en constante actualización y así mantener una relativa simetría con las nuevas técnicas, metodologías y tecnologías. Este constante procedimiento garantiza una mejor adaptabilidad de los estudiantes al medio laboral en que se desempeñarán.

Dentro del área de la Inteligencia Artificial (IA) el paradigma de desarrollo de agentes ha tenido un gran auge desde inicios del siglo. Esta nueva forma de ver la programación ha sido utilizada en varias áreas y dirige los principales desarrollos de sistemas inteligentes en la actualidad. En los entornos educativos se ha convertido en objeto de estudio en asignaturas de IA, pero sobre todo ha servido de base para el desarrollo de software educativo.

Introducción

Son varias las aplicaciones diseñadas con fines lúdicos o prácticos que se insertan en los cursos de Inteligencia Artificial (IA) en el mundo, sin embargo, muy pocas son creadas específicamente para la enseñanza. En la Universidad de las Ciencias Informáticas (UCI) se imparten dos asignaturas lectivas de IA. En estas materias se utiliza la herramienta *Swi-prolog* para la práctica del lenguaje *Prolog* y el *Weka* para ejecutar algoritmos de clasificación y agrupamiento en minería de datos. Estas herramientas no se ajustan a las necesidades de la formación, al menos su utilización directa, ya que no están dirigidas a desarrollar habilidades en la solución de problemas. Por otro lado en el tema Métodos de Solución de Problemas (MSP), los problemas de búsqueda heurística se ven desde una perspectiva teórica y las actividades prácticas se realizan de forma escrita. El estudiante debe formalizar los problemas mediante la definición de los elementos necesarios para ejecutar los algoritmos, sin embargo, la utilización de herramientas que permitan a los estudiantes programar o utilizar algoritmos ya programados en la solución de problemas computacionales es nula. Los estudiantes tienen que realizar los ejercicios en sus cuadernos o libretas, lo cual dificulta la comprobación de la eficiencia de la solución de los problemas, además, limita el desarrollo de habilidades prácticas de programación y solución de problemas.

De acuerdo a la situación antes descrita se plantea como **problema de investigación**: La necesidad de contribuir a la formulación y resolución práctica de problemas de búsqueda heurística en la asignatura Inteligencia Artificial 1.

La investigación tiene como **objeto de estudio**: La resolución de problemas de búsqueda heurística, quedando enmarcado el **campo de acción** en los sistemas agentes para la resolución de problemas de búsqueda heurística.

Para dar solución al problema planteado en la presente investigación se propone el siguiente **objetivo**: Desarrollar un agente resolvente de problemas de búsqueda heurística.

Para llevar a cabo el objetivo general se definieron las siguientes tareas de investigación:

1. Análisis histórico-lógico y síntesis del estado del objeto de estudio y el campo de acción.
2. Selección de las herramientas y metodología a utilizar en la implementación de la solución.
3. Análisis de la propuesta de solución.
4. Diseño de la propuesta de solución.
5. Implementación de un sistema agente para la resolución de problemas de búsqueda heurística.
6. Realización de las pruebas y documentación de los resultados.

Además, para todo el proceso de investigación y elaboración de este trabajo se tomará en cuenta la utilización de varios **métodos científicos de investigación** como:

- **Histórico-lógico**: Método que permitirá conocer los antecedentes y las tendencias actuales referidas a agentes y métodos de solución de problemas de búsqueda heurística, además de conceptos, términos y vocabularios propios del campo que contribuyen en gran medida al entendimiento del trabajo.
- **Analítico-sintético**: Mediante este método se podrá analizar teorías y métodos relacionados con los agentes y extraer los elementos más importantes de estos, los componentes por los que están conformados, su funcionamiento y la interacción entre ellos.
- **Modelación**: Utilizado para representar cómo ocurren los procesos que se desean automatizar, además para elaborar los diferentes modelos definidos en la metodología escogida y que sirven de guía durante el desarrollo de la solución.

Los métodos empíricos a utilizar para obtener información sobre el objeto de estudio son:

- **Consulta bibliográfica**: Permite la elaboración del marco teórico de la investigación fundamentada por la información consultada.
- **Pruebas de validación**: Confirma la veracidad y utilidad de la propuesta elaborada.

Introducción

El presente trabajo de diploma está estructurado de la siguiente forma: resumen, introducción, tres capítulos de contenido, conclusiones, recomendaciones, referencias bibliográficas y anexos. A continuación, se hace una breve descripción del contenido de cada uno de los capítulos:

Capítulo 1 "Fundamentación Teórica": se hace un análisis de los principales temas concernientes al objeto de estudio, se analizan los principales conceptos y términos asociados al problema en cuestión, así como otras soluciones existentes. Se refleja el estado del arte del tema de investigación y se seleccionan las herramientas y tecnología con que se desarrollará la solución propuesta.

Capítulo 2 "Solución propuesta": se realiza una descripción de los procedimientos seguidos de acuerdo a la metodología seleccionada para el proceso de desarrollo de *software*. Se muestran los artefactos generados en cada una de las etapas de desarrollo correspondientes a la estimación de tiempo, identificación de los requisitos del sistema y diseño de la lógica a implementar en el mismo.

Capítulo 3 "Pruebas e implementación del sistema": en este capítulo se definen los estándares de programación a utilizar, se desglosan las historias de usuario en tareas de ingeniería para organizar el trabajo de desarrollo y llevar a cabo una trayectoria de programación correcta. También se definen y realizan las pruebas al sistema para comprobar la correcta implementación de las funcionalidades orientadas.

Se espera con la presente investigación obtener una aplicación basada en el paradigma de agentes que permita a los estudiantes la formulación y resolución de problemas de búsqueda heurística. La característica de ser de código abierto permitirá servir de ejemplo y reutilizar su código para la resolución de problemas por parte de los estudiantes, lo cual debe contribuir a desarrollar sus habilidades y practicar los contenidos vistos en clases.

Capítulo 1: Fundamentación Teórica

Durante una investigación es necesario obtener conocimiento teórico que facilite la realización de las actividades y así poder llevar a la práctica lo aprendido. Para la presente investigación se hace necesario realizar una búsqueda bibliográfica en el tema de agentes y los Métodos de Solución de Problemas (MSP). Para ello se aborda en este capítulo los principales algoritmos de búsqueda heurística que conforman los MSP y que se utilizan en la asignatura Inteligencia Artificial 1. Por otro lado, se caracterizan las diferentes metodologías de desarrollo de software, el lenguaje de programación y el entorno de desarrollo a utilizar para la implementación del sistema que se propone.

1.2 Heurística

El significado de la palabra heurística, dentro del campo de la IA ha variado en la historia. En 1957, George Polya usó este término para referirse al estudio de métodos para descubrir e inventar técnicas de solución de problemas. En otras ocasiones se ha usado como un término opuesto a algorítmico. Por ejemplo, Newell, Shaw y Simon plantearon en 1993 que una heurística era *"Un proceso que puede resolver un problema dado, pero no ofrece garantía de hacerlo".* Actualmente, la heurística es frecuentemente usada como un adjetivo para referirse a cualquier técnica que mejore la media del proceso de solución de problemas (1).

Según Shapiro, uno de los resultados empíricos de los últimos treinta años de la Inteligencia Artificial es que para muchos problemas la relación o balance entre conocimiento, tiempo de cálculo y calidad de la solución es bastante favorable. Es decir, el uso de una pequeña cantidad de conocimiento específico del problema, puede mejorar significativamente la calidad de la solución o el costo del proceso de búsqueda. Esta es la esencia de la heurística.

Se define entonces una heurística como un conjunto de reglas que evalúan la posibilidad de que una búsqueda va en la dirección correcta (1).

La heurística no garantiza que siempre se tome la dirección correcta de una búsqueda, por eso este enfoque no es óptimo pero es mejor que los métodos de búsquedas a ciegas.

Un método heurístico es un procedimiento para resolver un problema de optimización bien definido, mediante una aproximación intuitiva en la que la estructura del problema se utiliza en forma inteligente para obtener una buena solución (2). Es un procedimiento de resolución de problemas para el que se tiene un alto grado de confianza en que se encuentren soluciones de alta calidad, con un costo computacional razonable, aunque no se garantice la optimalidad y en algunos casos ni la factibilidad.

1.3 Búsquedas heurísticas

Los métodos de solución de problemas en Inteligencia Artificial parten del término *heurística*. Generalmente los métodos de búsqueda heurísticas se basan en maximizar o minimizar algunos aspectos del problema. Están orientados a reducir la cantidad de búsqueda requerida para encontrar una solución.

Cuando un problema es presentado como un árbol de búsqueda, el enfoque heurístico intenta reducir el tamaño del árbol cortando nodos poco prometedores.

La búsqueda heurística es un método de búsqueda que se basa en reglas “tipo suerte”, estas reglas, por lo general, resultan exitosas, aunque su éxito no está garantizado. Supone la existencia de una función de evaluación $h(n)$ que debe medir la distancia estimada al objetivo (representada de forma abstracta mediante nodos). Esta función de evaluación se utiliza para guiar el proceso, hace que en cada momento se seleccione el estado o las operaciones más prometedoras, aunque no siempre se garantiza encontrar una solución en caso de que exista, ni de encontrar la más próxima. Estos algoritmos de búsqueda dependen de la calidad de la función heurística que se utiliza para calcular la distancia desde el nodo actual hasta el nodo objetivo (3).

La función de evaluación heurística es una función que hace corresponder situaciones del problema con números. Es decir, brinda una medida conceptual de la distancia entre un estado dado y el estado objetivo. Estos valores son usados para determinar cuál operación ejecutar a continuación, típicamente seleccionando la operación que conduce a la situación con máxima o mínima evaluación.

Aunque existen infinidad de métodos de búsqueda, se pueden identificar dos grandes grupos:

- Los dedicados a encontrar el camino de un estado inicial a un estado objetivo.
- Los que transforman un estado, a partir de un conjunto de operadores, para encontrar otro estado que cumpla en mayor grado con una función de evaluación. Estos métodos generalmente son llamados métodos de búsqueda local.

En la presente investigación se analizan los principales algoritmos de búsqueda heurística impartidos en clases a los estudiantes de la carrera de Ingeniería en Ciencias Informáticas.

1.4 Algoritmos de búsqueda local

Estos métodos se han aplicado a diferentes problemas de optimización fundamentalmente, aunque también se han utilizado en problemas de planificación, diseño y otras áreas.

1.4.1 Algoritmo Escalador de Colinas

Escalador de colinas se emplea sobre todo para resolver problemas de optimización, como por ejemplo, encontrar la secuencia óptima para procesar un conjunto de tareas por un computador (4).

Algunas de las características más importantes de este algoritmo son:

Informado: Utiliza información del estado actual y el posible próximo estado para elegirlo o no.

No exhaustivo: No explora todo el espacio de estados. Como máximo, sólo encuentra una solución. Encuentra buenas soluciones, pero no la mejor, puesto que no es exhaustivo.

Eficiente: Evita la exploración de una parte del espacio de estados que puede no ser prometedora.

Otro elemento de interés es que es útil si se tiene una función heurística muy buena o cuando los operadores de transición entre estados tienen cierta independencia (conmutativa), que implica que la operación de un operador no altera la futura aplicación de otro. Además, no guarda los estados anteriores, lo cual permite una buena eficiencia en memoria. Dado su bajo costo computacional es una de las estrategias de búsqueda más utilizada en optimización y aprendizaje. Sin embargo, cuando existen "colinas falsas" ocurre que el algoritmo se detiene debido a que todos los nodos futuros pueden parecer peores que el actual.

Para buscar en un espacio dado, utilizando el algoritmo mencionado, la estrategia base que se aplica es hacer una búsqueda dirigida empleando una función $f(x)$ que permita obtener una medida estimada de la distancia a la que se encuentra el nodo objetivo dado un nodo (3).

Este algoritmo tiene muchas variantes, una de las más básicas se presenta a continuación:

- Formar una pila de un elemento consistente del nodo raíz.
- Iterar de (a) a (c) hasta que la pila este vacía.
  a) Chequear si el elemento del tope de la pila es una solución.
  b) SALIR con ÉXITO si el elemento chequeado es solución.
  c) Si el primer elemento no es solución ordenar los descendientes del mismo según la función $f(x)$ y luego añadir los que son mejores que el actual.
- Salir con Falla.

La siguiente imagen representa el pseudocódigo de ejecución del algoritmo escalador de colinas.

```
Entrada: PilaDeEstados
Salida: Retorna el estado final EstadoActual
1: EstadoActual := POP(PilaDeEstados)
2: mientras EstadoFinal! = EstadoActual hacer
3:    Hijos := generarSucesores(EstadoActual)
4:    Hijos := ordenarYeliminarPeores(Hijos, EstadoActual)
5:    si no vacio(Hijos) entonces
6:       EstadoActual := EscojerMejor(Hijos)
7:    fin si
8: fin mientras
9: devolver EstadoActual
```

Ilustración 1. Pseudocódigo del algoritmo escalador de colinas (Elaboración Propia).

La escalada de colinas mejora gradualmente una solución seleccionando al mejor vecino basándose en una función de evaluación, hasta que no haya un vecino mejor que el actual. Si hay más de un mejor sucesor, se selecciona un nodo del conjunto de los mejores sucesores. Pero este algoritmo no es completo, pues usualmente cae en ciclos como máximos locales donde los vecinos del nodo actual no son mejores que este. Atendiendo a esta situación, es necesario tomar decisiones como comenzar nuevamente el algoritmo desde otro nodo de partida o forzar la búsqueda seleccionando un nodo aleatorio desde el cual seguir aplicando escalador de colinas.

1.4.2 Algoritmo Recocido Simulado

Una de las técnicas reconocidas para atenuar las deficiencias del escalador de colinas es aplicar la filosofía de enfriamiento de metal. Cuando un material se enfría lentamente pierde su energía y finalmente en un punto del tiempo llega al estado de mínima energía. De la observación de este fenómeno se puede deducir que los procesos físicos tienen transiciones entre estados altos y bajos de energía; luego este pensamiento puede aplicarse al estado de máximo local del algoritmo escalador de colinas y hacerlos saltar a un próximo estado (6). Esta vía de escape es conocida como el algoritmo Recocido Simulado.

Recocido Simulado se basa en aceptar en forma limitada transiciones que no mejoren la función de costo usando mecanismos no deterministas. Este establece una conexión entre este tipo de proceso termodinámico y la búsqueda de un mínimo global. Además, es un algoritmo de aproximación a la solución óptima, fundado en una analogía con el comportamiento de sistemas termodinámicos (7).

Es un algoritmo de mejora iterativa en que la aleatoriedad se incorpora para ampliar el espacio de búsqueda y evitar quedar atrapado en un mínimo local (6).

La probabilidad de que un cambio aleatorio de estado suceda es muy pequeña y está dada por:

$$p = e^{-\Delta E/kT}$$

Donde:

- *p* es la probabilidad de transición de un bajo a un alto estado de energía.
- *k* es la constante de *Boltzman*.
- *T* es la temperatura en el estado en curso.
- ΔE denota un cambio positivo de la energía.

Nótese que para un ΔE pequeño, la probabilidad es alta, y viceversa.

El proceso es gobernado por el parámetro T. El plan de cómo variar T se denomina esquema de recocido. Un enfriamiento demasiado rápido puede llevar a regiones estables de alta energía (mínimos locales en lugar de globales). Un enfriamiento lento propicia alcanzar con más seguridad una estructura uniforme (mínimo global), pero si es demasiado lento se pierde tiempo.

El pseudocódigo del recocido simulado se muestra en la siguiente imagen.

Salida: Retorna el estado final S
1: $S := GeneraUnaSolucionInicial()$
2: $T := T_0$; $g := 0$
3: **mientras** $CondicionesDeParoNoActivas(g, T)$ **hacer**
4: $S := TomaUnVecinoAleatorioDe(S)$
5: **si** $(f(S) < f(S'))$ ó $(Random(0, 1.0) > \frac{f(S)-f(S')}{T})$ **entonces**
6: $S := S$
7: **fin si**
8: $g := g + 1$
9: $T := Actualiza(g, T)$
10: **fin mientras**
11: **devolver** S

Ilustración 2. Pseudocódigo del Algoritmo Recocido Simulado (Elaboración Propia).

Como se puede apreciar este algoritmo es un escalador de colina que incorpora elementos de aleatoriedad para poder escoger estados peores que el actual. En la presente investigación el objetivo es implementar algoritmos básicos de ejemplo para ser modificados por los estudiantes, por lo que se considera el escalador de colinas el indicado para incorporar al software.

1.5 Algoritmos de búsqueda de caminos

Esta familia de algoritmos crea un árbol de búsqueda a través de todo el espacio de estados. Van guardando información de todo el camino seguido desde el estado inicial hasta el estado objetivo. Esta característica le permite ser útil en problemas de rutas, planificación y recorridos, entre otros.

1.5.1 Algoritmo primero el mejor

En la búsqueda Primero el Mejor, el espacio de búsqueda es determinado a partir de una función de evaluación $f(n)$. Esta función consta de dos partes, una función heurística $h(n)$ y el costo estimado $g(n)$ donde:

$$f(n) = g(n) + h(n)$$

El algoritmo básico primero el mejor, asume el costo $g(n)$ como 0, es decir, no toma en cuenta el costo desde el estado inicial hasta el estado actual.

Una de las formas más simples para obtener $h(n)$ es estimar el costo para alcanzar el objetivo desde el nodo que se evalúa. Es decir la estrategia es expandir primero el nodo cuyo estado se valora como más próximo al estado objetivo. Por ejemplo, la distancia aérea o en línea recta, a partir de un mapa, desde el nodo actual al nodo objetivo.

De esta forma se selecciona un nodo inicial y se comienza a aplicar el algoritmo. Primero se compara el nodo actual con el nodo objetivo y si no es el objetivo, se expanden sus nodos adyacentes y se calculan sus funciones de evaluación, luego se ingresan en una cola con prioridad donde el menor valor es el más priorizado. Luego se extraen los nodos aplicando la misma estrategia hasta encontrar el nodo objetivo o vaciar la cola.

La búsqueda primero el mejor, frecuentemente trabaja bien. Tiende a encontrar soluciones rápidamente, aunque no siempre encuentre la óptima. Tampoco es una búsqueda completa.

En la siguiente imagen se muestra el pseudocódigo del método general Primero el Mejor.

Entrada: Procedimiento $Backtracking(X[1...i] : Resultado, OK : Booleano)$.
Lista de nodos iniciales $ListaDeEstados$
Salida: Retorna el camino C

```
1: si EsSolucion(X) entonces
2:    OK := true
3: si no
4:    OK := False
5:    L := Candidatos(X)
6:    mientras (no OK) y (no vacio(L)) hacer
7:       X[i + 1] := Cabeza(L), L := Resto(L)
8:       Backtracking(X, OK)
9:    fin mientras
10: fin si
```

Ilustración 3. Pseudocódigo del Algoritmo Primero el Mejor (Elaboración Propia).

1.5.2 Algoritmo A*

El algoritmo Primero el Mejor toma en cuenta los valores de la heurística para llegar al nodo objetivo, pero no valora todo lo que le costó llegar hasta su posición actual. Por tanto, se decide utilizar el costo hasta donde ha llegado, lo que da paso al algoritmo A*.

A* ejecuta el mismo procedimiento del algoritmo Primero el Mejor, lo único que cambia es que ahora la ordenación de los nodos se realiza utilizando el valor de $f(n)$ completo; que a igual valor de $h(n)$ los nodos con $g(n)$ más pequeñas se explorarán antes. Dado que el objetivo de este algoritmo no es llegar lo más rápidamente a la solución, sino encontrar la de menor coste, se tiene en cuenta el coste de todo el camino y no solo el camino por recorrer (8).

Cabe destacar que el algoritmo solo acaba cuando se extrae una solución de la cola. Puede que en algún momento exista en la estructura de nodos abiertos, nodos solución, pero hasta que no se hayan explorado los nodos por delante de ellos, no se puede asegurar que realmente sea una solución buena. Se podrá obtener una solución óptima para un problema siempre que se tenga en cuenta que el coste espacial del algoritmo A* crece exponencialmente a no ser que cumpla que:

$$|h(n) - h^*(n)| < O(\log(h^*(n)))$$

Donde:

- $h^*(n)$: Costo mínimo desde un nodo cualquiera hasta el nodo objetivo.
- $h(n)$: Función heurística.
- $O(\log(h^*(n)))$: Complejidad algorítmica.

El algoritmo A* puede verse como el exponente más completo dentro de los algoritmos primero el mejor, ya que toma en cuenta la heurística y el costo.

Seguidamente se muestra el pseudocódigo del algoritmo mencionado.

```
Entrada: EstadoInicial
Salida: Retorna el estado final EstadoActual
1: EstadosAbiertos.insertar(EstadoInicial)
2: EstadoActual := EstadosAbiertos.primero()
3: mientras (no esFinal(EstadoActual)) y (no EstadosAbiertos.vacio())
   hacer
4:    EstadosAbiertos.borrarPrimero()
5:    EstadosCerrados.insertar(EstadoActual)
6:    Hijos := generarSucesores(EstadoActual)
7:    Hijos := tratarRepetidos(Hijos, EstadosCerrados, EstadosAbiertos)
8:    EstadosAbiertos.insertar(Hijos)
9:    EstadoActual := EstadosAbiertos.primero()
10: fin mientras
11: devolver EstadoActual
```

Ilustración 4. Pseudocódigo del Algoritmo A(Elaboración Propia).*

El algoritmo A* tiene sus limitaciones de espacio, impuestas principalmente por la posibilidad de acabar degenerando en una búsqueda en anchura si la función heurística no es demasiado buena. Además, si la solución del problema se encuentra a un nivel de profundidad elevado o el tamaño del espacio de búsqueda es muy grande, resulta inevitable llegar a necesidades de espacio prohibitivas. Esta deficiencia proclama a que sea objetivo plantearse algoritmos alternativos con menores necesidades de espacio.

1.5.3 Algoritmo IDA*

El algoritmo A* admite varias optimizaciones, como no aumentar el coste de uno en uno, sino averiguar cuál es el coste más pequeño de los nodos no expandidos en la iteración actual y usarlo en la siguiente iteración. El bucle interior es el que realiza la búsqueda en profundidad limitada y también se podría optimizar utilizando una implementación recursiva.

El algoritmo IDA*, necesita una cantidad de espacio lineal, lo cual permite hallar soluciones a más profundidad, sin embargo se debe pagar como precio el aumento de espacio y consumo de recursos al tener que re-expandir los nodos ya visitados. Este

consumo extra dependerá de la conectividad del grafo en el espacio de estados. Si existen muchos ciclos este puede ser relativamente alto ya que en cada iteración, la efectividad de la exploración real se reduce con el número de nodos repetidos que aparecen.

Entrada: $EstadoInicial$
Salida: Retorna el estado final $EstadoActual$

```
1:  profundidad := f(EstadoInicial)
2:  EstadoActual := EstadosInicial
3:  mientras (no esFinal(EstadoActual)) y (profundidad < limite) hacer
4:      EstadosAbiertos.inicializa()
5:      EstadosAbiertos.insertar(EstadoInicial)
6:      EstadoActual := EstadosAbiertos.primero()
7:      mientras (no esFinal(EstadoActual)) y (no EstadosAbiertos.vacio())
        hacer
8:          EstadosAbiertos.borrarPrimero()
9:          EstadosCerrados.insertar(EstadoActual)
10:         Hijos := generarSucesores(EstadoActual, profundidad)
11:         Hijos := tratarRepetidos(Hijos, EstadosCerrados, EstadosAbiertos)
12:         EstadosAbiertos.insertar(Hijos)
13:         EstadosActual := EstadosAbiertos.primero()
14:     fin mientras
15:     profundidad := profundidad + 1
16: fin mientras
17: devolver EstadoActual
```

Ilustración 5. Pseudocódigo del Algoritmo IDA (Elaboración Propia).*

Este algoritmo es una modificación del A* como otras tantas que buscan optimizar espacio en memoria o tiempo, tratando de afectar lo menos posible el resultado. Sin embargo, el algoritmo A* es el que ofrece una solución óptima y es el algoritmo base, a partir del cual se derivan los

demás. Este es el elemento de mayor importancia para ser incorporado al software para la resolución de problemas en la asignatura Inteligencia Artificial 1.

1.6 Agente inteligente

Antes de describir este concepto es necesario conocer qué es un agente, según Russell, en (3), un agente "*es cualquier ente capaz de percibir su medio ambiente con la ayuda de sensores y actuar en el medio, utilizando actuadores*[1]".

Las cualidades que tendría un agente inteligente ideal, según James A. Hendler en (9), serían:

- **Comunicativo**: el agente debe entender las necesidades, objetivos y preferencias del usuario para que este pueda realizar su función correctamente.
- **Capaz**: el agente no sólo debe proporcionar una información, sino también un servicio, es decir, debe tener capacidad para hacer cosas.
- **Autónomo**: el agente, además de comunicarse, debe poder interactuar con el entorno, tomando decisiones y actuando por sí solo, limitando sus acciones según el nivel de autonomía permitida por el usuario.
- **Adaptativo**: debe ser capaz de aprender del entorno: preferencias de usuarios, fuentes de información y de otros agentes.

Es necesario, para una mejor comprensión de lo que es un agente inteligente conocer de sus principales características. Los agentes inteligentes se han definido de diferentes maneras y, aunque no necesariamente tienen que cumplir todas las características, sí intentan abarcar la mayor cantidad posible:

- Aprender nuevos problemas e incrementar normas de solución.
- Capacidad de adaptación en línea y en tiempo real.
- Ser capaz de analizar condiciones en términos de comportamiento, el error y el éxito.
- Aprender y mejorar a través de la interacción con el medio ambiente (realización).
- Aprender rápidamente de grandes cantidades de datos.

[1] Se usa este término para indicar el elemento que reacciona a un estímulo realizando una acción.

- Deben estar basados en memoria de almacenamiento masivo y la recuperación de dicha capacidad.

Estructura del agente.

Se reconoce en la bibliografía la estructura básica de los agentes, formada por la percepción, las acciones, un objetivo básico bien definido y el entorno donde se desenvuelve. Asumiendo un agente como una entidad en un entorno virtual como un videojuego, se puede caracterizar cada uno de estos elementos de la siguiente manera:

- **La capacidad de percepción**: viene definida por los elementos capaces de reconocer de los que dispone el agente. Sistemas sencillos en los que la percepción puede ser la detección o no de intrusos en su área de acción (definida fácilmente con un booleano) o por mecanismos más complejos como una matriz de NxM que refleje la visión del agente en una orientación y momento concreto del tiempo y que requerirá un proceso más intenso e incluso una abstracción para agilizar cálculos.
- **La capacidad de acción**: Vendría definida por el conjunto de los movimientos, cálculos o respuestas en general que puede llevar a cabo el agente. Pueden ser tan sencillos como (giro izquierda/giro derecha/avanzar/retroceder) o más complejos como (evadir/emboscar/atacar/confundir).
- **Los objetivos**: Son la esencia del agente. El comportamiento del mismo irá orientado a la consecución de los mismos.
- **El entorno**: Es una característica externa al agente pero que condiciona su comportamiento. Puede ser un mundo tridimensional o una abstracción del mismo reducida a eventos. En otros casos puede ser una matriz la que modele el entorno o incluso un grafo que represente una topología concreta.

Teniendo en cuenta la variedad que pueden presentar los agentes, estos pueden ser clasificados en grupos en correspondencia con el comportamiento que estos desempeñen. Así se pueden considerar agentes simples, que resuelven problemas

específicos o sistemas multiagentes, donde interactúan varios de ellos de forma colaborativa para resolver problemas más complejos.

En la presente investigación se defiende la idea de un agente simple que cumpla las características básicas para resolver problemas de búsqueda heurística.

1.7 Herramientas, tecnología y metodología de desarrollo

Todo desarrollo de software se realiza a partir de un análisis de las tecnologías y herramientas que mejor se adapten para la implementación del mismo. Sin dejar de ejecutar todos los procedimientos bajo una guía o metodología de desarrollo que garantice un correcto proceso de construcción del producto.

1.7.1 Lenguaje de programación

Una computadora funciona bajo control de un programa, el cual debe estar almacenado en la unidad de memoria; tales como el disco duro. Los lenguajes de programación de una computadora en particular se conocen como código de máquinas o lenguaje de máquinas, mientras que los lenguajes de programación de software son herramientas que permiten crear programas computacionales que ejecutan las computadoras. Existen diversos lenguajes de programación que podrían ser utilizados para la implementación de la solución, atendiendo a esta diversidad se realizó una consulta al índice de TIOBE. El índice TIOBE (TIOBE, *The Importance of Being Earnest*) es un informe mensual que elabora y publica la empresa *TIOBE Software BV*, solo se basa en la cantidad de líneas de código utilizadas por los programadores de todo el mundo (33). Atendiendo a los análisis de esta organización, el lenguaje más utilizado en los últimos años es Java, que actualmente se encuentra en su versión 8. La siguiente imagen muestra una comparación de la utilización de los 10 primeros lenguajes en la lista confeccionada por esta institución.

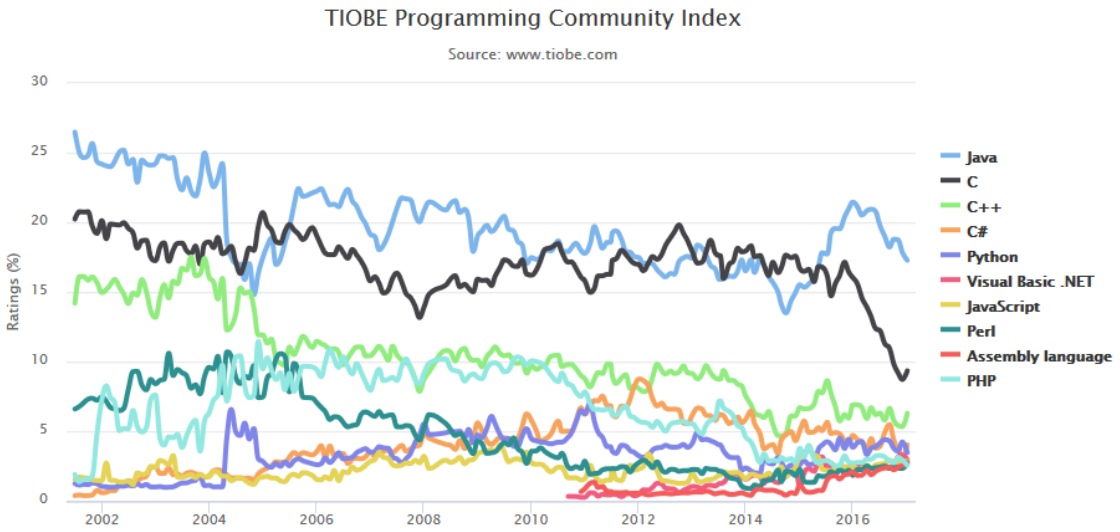


Ilustración 6. Utilización de lenguajes de programación según el índice de TIOBE (Tomado de https://www.tiobe.com/tiobe-index/).

También se tienen en cuenta algunas características del propio lenguaje que lo hacen ideal para su utilización en la presente investigación. Es un lenguaje de programación de propósito general, multiplataforma, concurrente, orientado a objetos y basado en clases (9).

Cabe destacar el peso que tiene el lenguaje Java en la enseñanza en la UCI, es el lenguaje con que aprenden los estudiantes, lo cual lo hace muy favorable para implementar el sistema propuesto. Además, es muy utilizado en los programas auxiliares que se utilizan para la enseñanza de la asignatura IA en la UCI, así como para la implementación de agentes inteligentes como es reflejado en (Sistema inteligente para la recomendación de objetos de aprendizaje, (10)), (Un sistema de tutoría inteligente adaptativo considerando estilos de aprendizaje, (11)) y (Sistema de vigilancia de personas mayores o con incapacidad usando sistemas multiagente y robots bípedos controlados mediante voz, (12)).

Para garantizar un trabajo ágil en la implementación de la propuesta de solución, se considera la utilización de algún entorno de desarrollo que influya en el ahorro de tiempo y reduzca la probabilidad de cometer errores de programación.

1.7.2 Análisis de Entornos de Desarrollo Integrados

Un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse de forma exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios. Una decisión importante a la hora de desarrollar con Java es la selección del IDE, ya que el entorno de desarrollo que se elija puede suponer una verdadera diferencia en el tiempo de trabajo invertido. Debido a lo anterior se realiza un estudio sobre algunos de los IDE diseñados para trabajar con Java.

- **NetBeans**: es multiplataforma (Windows, Linux, Mac OS X y Solaris), gratuito, de código abierto (con licencia CDDL). Esta herramienta permite el desarrollo rápido y fácil de aplicaciones Java de escritorio, móviles y aplicaciones web. Ofrece soporte para las últimas tecnologías Java. A parte de las funciones básicas con las que debería contar cualquier IDE, como resaltado de sintaxis, autocompletado, formateo de código o depurador (*xDebug*), también cuenta con otras funcionalidades menos comunes como la integración con *CVS*, *Subversion* y *Mercurial* para el control de versiones (13). Además es objeto de colaboración de una gran comunidad de usuarios y desarrolladores de todo el mundo.
- **Eclipse**: es una herramienta multiplataforma compuesto por un conjunto de herramientas de programación de código abierto, combina soporte para los lenguajes Java, C/C++ y PHP, contando con una gran gama de *plugins* que lo hacen altamente personalizable. Dispone de un editor de texto con un analizador sintáctico. La compilación es en tiempo real, permite la realización de pruebas unitarias con *JUnit*, control de versiones con CVS. Asimismo, a través de *plugins* libremente disponibles es posible añadir control de versiones con *Subversion* así como llevar a cabo una integración con *Hibernate* (14). Existe una gran comunidad de usuarios y desarrolladores que dan soporte a esta herramienta.
- **IntelliJ IDEA**: es un IDE para el desarrollo de programas informáticos que cuenta con licencia comercial y con edición comunitaria. Ofrece soporte para una gran cantidad de lenguajes entre ellos *Java* 8 y contiene una interfaz de usuario para el desarrollo de *Android*. Permite el control de versiones integrado con *Git* y brinda

un análisis sintáctico del lenguaje que se implemente apoyado además por un completado inteligente de código (15).

Al considerar algunas características de estos entornos se pudo observar que *Eclipse* necesita de varios *plugin* para poder obtener el resultado que se desea, mientras que *IntelliJ IDEA* ofrece un muy amplio espectro de desarrollo por lo que demanda altas prestaciones para su correcta ejecución. En tanto *NetBeans* es el que mejor se presenta para la situación que se pretende resolver. Se hace necesario destacar que una ventaja de este IDE es su integración con la plataforma de desarrollo *JavaFX*. Esta es una plataforma pensada para crear y desplegar aplicaciones para internet que funcionan en una gran cantidad de dispositivos (escritorio, navegadores y móviles) y utiliza Java como lenguaje de programación. Esta plataforma provee más de 50 componentes de diseño, formularios fácilmente personalizables mediante el uso de Hojas de Estilo (CSS, por sus siglas en inglés). Además ofrece al programador grandes ventajas debido a que posee un motor de renderizado web y un motor de multimedia de alto rendimiento (16). Atendiendo a las características mencionadas de esta última herramienta, se utilizará *JavaFX* 2.2 para apoyar el desarrollo visual de la propuesta de solución.

Una vez seleccionadas las herramientas y tecnologías con que se trabajará, se hace necesario establecer una guía para llevar a cabo el desarrollo, por lo que resulta esencial establecer una planificación que garantice una correcta implementación de las funcionalidades con que contará la solución. Para ello, se han definido internacionalmente un conjunto de metodologías de desarrollo de software que proponen las pautas, mecanismos de control y la documentación a realizar para llevar a cabo la ejecución de un proyecto.

1.7.3 Metodología de desarrollo de software

Una metodología es el conjunto de fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de un sistema (17). En ingeniería de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. El desarrollo de software es un proceso difícil y lleno de riesgos donde la metodología que se aplique para realizar

el mismo, define quién debe hacer qué, cuándo y cómo debe hacerlo. El fracaso en el desarrollo de un sistema está estrechamente relacionado con una mala selección de la metodología (18), por lo que este proceso se debe realizar a partir de un análisis previo y exhaustivo por parte del equipo de desarrollo. De acuerdo al corto tiempo disponible para la implementación de la solución, se analizan un grupo de metodologías ágiles, las cuales son descritas a continuación.

- **Scrum**: La metodología *Scrum* es un modelo de referencia que define un conjunto de prácticas y roles que pueden tomarse como punto de partida para el proceso de desarrollo de software. Su principal característica es que el desarrollo de software se realiza mediante iteraciones denominadas sprint, donde cada sprint representa un incremento del producto. En tanto, a lo largo del proyecto se llevan a cabo reuniones diarias para la coordinación e integración de las actividades a desarrollar (19). Un principio clave de *Scrum* es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y necesitan, y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, esta metodología adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes (20). Aunque *Scrum* se enfoca en la gestión de procesos de desarrollo de software, puede ser utilizado en equipos de mantenimiento de software.
- **Programación Extrema**: Programación Extrema (XP, por sus siglas en inglés) es la más destacada de los procesos ágiles de desarrollo de software. Tiene éxito porque hace hincapié en la satisfacción del cliente. Faculta a sus desarrolladores para responder con seguridad a las necesidades cambiantes de los clientes, incluso en etapas tardías del ciclo de vida del producto. Enfatiza el trabajo en equipo. Los jefes de proyecto, clientes y desarrolladores son socios iguales en un equipo de colaboración que permite la reorganización en torno al problema a resolver de la forma más eficiente posible. Los programadores extremos se comunican constantemente con sus clientes y colegas de trabajo, mantienen su

diseño simple y limpio (21). Cada pequeño éxito profundiza su respeto por las contribuciones únicas de cada uno y cada miembro del equipo. Con esta base los programadores extremos son capaces de responder con valentía a las cambiantes necesidades y la tecnología.

- **Proceso Unificado Ágil**: también conocido por sus siglas en inglés AUP (*Agile Unified Proccess*) es una metodología fácil de entender para el desarrollo de aplicaciones software de negocio, utilizando técnicas ágiles y conceptos aun fieles a las del *Rational Unified Proccess* (RUP), por lo tanto, es una versión simplificada de RUP. Posee muchas de las técnicas ágiles de las que dispone la metodología XP y de las formalidades de RUP, teniendo como filosofía adaptarse a las necesidades del proyecto. También, plantea un ciclo de vida iterativo, que se basa en la ampliación y refinamiento sucesivo del sistema, mediante múltiples iteraciones con retroalimentación cíclica y adaptación como elementos principales que dirigen para converger hacia un sistema adecuado (22).

Teniendo en cuenta las características mencionadas de las metodologías señaladas anteriormente, se selecciona XP como la que mejor se adapta a las condiciones de trabajo y las metas a cumplir en la presente investigación durante el corto tiempo de desarrollo del que se dispone. Además, XP es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y retroalimentación o reutilización del código desarrollado. Esta metodología se caracteriza por centrarse en fortalecer las relaciones interpersonales para el éxito del desarrollo de software, promueve el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen ambiente de trabajo. También cabe destacar su utilización para el desarrollo de agentes inteligentes como se puede ver en (23), (24), (25) y (26).

1.8 Conclusiones del capítulo

A partir del análisis de la bibliografía consultada y los estudios de algoritmos realizados, se puede concluir que:

- Las características analizadas de los algoritmos A* y Escalador de Colinas permitieron seleccionarlos como los mejores candidatos para establecer el

sistema base que resolverá los problemas de búsqueda de caminos y búsqueda local respectivamente.

- La gran utilización en el ámbito internacional, así como ser el lenguaje base de enseñanza en la UCI, permitió seleccionar al Java como lenguaje para el desarrollo del sistema. A partir de esta selección se escoge el IDE *NetBeans* en su versión 8.2 y el *framework JavaFX* para el desarrollo de interfaces visuales.
- Para llevar a cabo el proceso de ingeniería de software se acoge XP como metodología de desarrollo, se toma en cuenta para ello que garantiza la generación de los artefactos necesarios para una mayor comprensión de la solución a desarrollar.

Capítulo 2: Propuesta de Solución

En el presente capítulo se realiza una descripción del alcance que tiene la propuesta de solución permitiendo tener una concepción general del sistema. En el mismo, se definen y modelan los procesos del negocio que intervienen, se expone una representación de la lógica del sistema donde se representan los conceptos fundamentales del dominio del problema y se identifican y definen los requisitos funcionales y no funcionales de la aplicación. Además, se especifican las condiciones y pasos a tener en cuenta para el correcto funcionamiento de la solución.

2.1 Descripción de la solución

Los elementos básicos de la definición de un problema son los estados en los cuales se representa el problema y las acciones que permiten transitar de un estado a otro. De forma general los elementos que incluyen los problemas de búsqueda son los siguientes:

- El estado inicial donde se encuentra el agente.
- El conjunto de acciones u operadores posibles disponibles al agente. El término operador se usa para denotar la descripción de una acción en términos de cual estado será alcanzado ejecutando la acción en un estado particular. Una formulación alternativa es usar una función sucesor S; dado un estado particular x, S(x) retorna al conjunto de estados alcanzables desde x mediante una acción simple.
- El espacio de estados del problema es el conjunto de todos los estados alcanzables a partir del estado inicial mediante una secuencia de acciones cualquiera.
- Un camino en el espacio de estado es una secuencia de acciones que conduce de un estado a otro. Este elemento es de interés para los algoritmos de búsqueda de caminos.
- El criterio objetivo es el criterio que el agente usa para aplicarlo a la descripción de un estado para determinar si este es un estado objetivo (lo que se desea obtener). Algunas veces hay un conjunto explícito de posibles objetivos y el criterio

simplemente consiste en chequear si se ha alcanzado uno de ellos. Otra variante es especificar el objetivo por una propiedad abstracta, por ejemplo, el criterio de jaque mate del ajedrez.

- El costo de un camino es una función que asigna un costo a un camino.
- Una solución para los algoritmos de búsqueda de caminos es un camino desde el estado inicial a un estado que satisface el criterio objetivo. Para el caso de los algoritmos de búsqueda local es un estado que cumple el criterio objetivo.

La presente investigación pretende obtener como resultado un sistema inteligente que tendrá integrado un agente, este sistema debe ser capaz de resolver problemas en dependencia de los elementos o parámetros definidos por el usuario del sistema. Inicialmente debe ser posible introducir los estados del problema, los valores de la heurística y la matriz de costo. Una vez establecidos los datos del problema a resolver, el agente seleccionará el algoritmo adecuado a aplicar para encontrar la solución deseada.

El software se encargará de todo el procesamiento lógico necesario para llegar a una solución determinada. A partir de los resultados obtenidos al aplicar uno u otro algoritmo, el usuario tiene la posibilidad de cuestionar y verificar la eficiencia de cada uno, de acuerdo al problema modelado. Se desea que el sistema sea de código abierto, para que los estudiantes modifiquen los algoritmos o puedan reutilizar el código para desarrollar otras aplicaciones. Como forma de introducir el tema en la enseñanza y las ventajas presentadas en el epígrafe 1.6, será implementado siguiendo un modelo de agente inteligente que permitirá el razonamiento artificial. Para lograr un resultado acorde al esperado se necesita seguir los patrones arquitectónicos relacionados con los agentes, aspecto tratado en el siguiente epígrafe.

2.2 Patrón arquitectónico

Uno de los modelos más antiguos y que mantiene vigencia en el desarrollo de agentes es el que asume 3 componentes esenciales en el agente: Las creencias, Los deseos y

las Intenciones (BDI, por sus siglas en inglés). Este brinda un diseño genérico para el desarrollo de un agente inteligente. Está basado en el modelo cognitivo del ser humano, de modo que los agentes utilizan una representación interna de su entorno, un modelo del mundo que los rodea.

Un agente percibe a través de sus sensores una serie de estímulos procedentes de su entorno, los cuales modifican el modelo del mundo que tiene el agente o sea sus creencias (*Beliefs*) acerca del mundo. El agente tiene que tomar decisiones para interactuar con el entorno, las cuales estarán basadas en sus creencias y gobernadas por sus deseos (*Desires*). Estos deseos se interpretan como objetivos que el agente quiere alcanzar y ha de valerse de una serie de intenciones (*Intentions*) para lograrlo. Por tanto, las intenciones no son otra cosa que acciones que el agente va realizando, las cuales podrían ser abortadas en cualquier momento si las creencias del agente cambian.

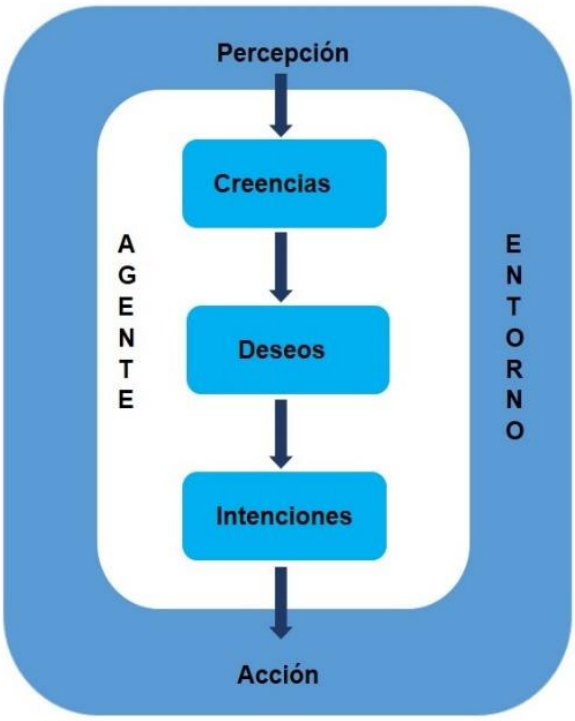


Ilustración 7. Modelo BDI.

El agente a desarrollar basará sus creencias en los elementos que caracterizan a los algoritmos y en las percepciones que realiza en su entorno a través de la interfaz.

Permitirá la construcción de un árbol de búsqueda y asignará la heurística que le corresponde a cada estado.

Para cumplir con los objetivos del agente, este debe valerse de intenciones que no son más que las acciones que realizará el agente en su entorno; estas son: llevar a cabo el algoritmo y mostrar mensajes de información al usuario con la solución. Además, forman parte de sus intenciones cada uno de los predicados intermedios a implementar para crear el árbol de búsqueda. Luego de haber estudiado este modelo se definió una estructura general del sistema.

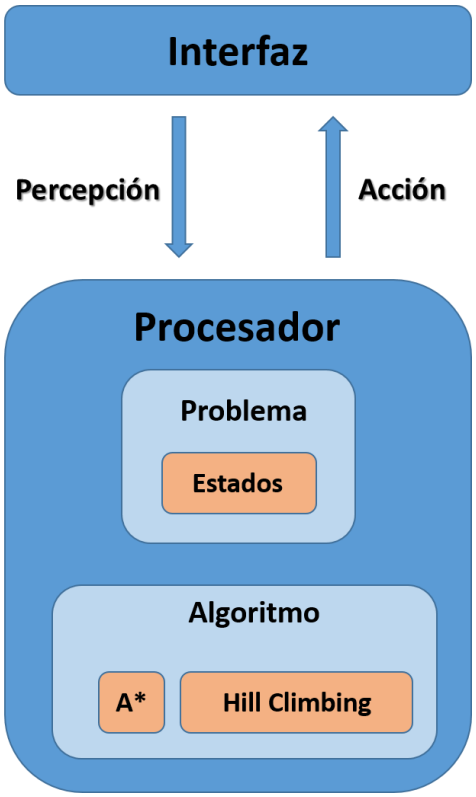


Ilustración 8. Estructura del sistema.

En la Ilustración 8. Estructura del sistema. se muestra que el sistema cuenta con una capa interfaz la cual será la que permita la interacción del usuario con el mismo. Esta capa se comunica con el agente a través de la clase Percepción, indicando los datos del problema que desea resolver el usuario. Luego, la clase Procesador identifica el tipo de problema, el cual contiene el conjunto de estados que lo definen, para conocer los datos

del mismo y determinar qué algoritmo debe aplicar. Luego que el agente realiza las operaciones necesarias, este le comunica al usuario el resultado de la ejecución del algoritmo previamente seleccionado a través de la clase Acción. Al definir la estructura del sistema y el patrón arquitectónico, el desarrollo del proyecto se encuentra preparado para establecer los requerimientos del software.

2.3 Especificación de los requisitos de software

Todo sistema de software debe cumplir con un conjunto de condiciones o capacidades que resultan ser las restricciones bajo las cuales debe operar. Debe lograrse un entendimiento entre el equipo de desarrollo y el cliente, y especificar las necesidades reales de forma que cumpla con sus expectativas.

2.3.1 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. En muchos casos los requisitos no funcionales son fundamentales en el éxito del producto. Normalmente están vinculados a requisitos funcionales, es decir, una vez se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser (18). Seguidamente se definen las principales características que representarán al sistema.

Apariencia o interfaz externa

- Interfaces uniformes y con los mismos colores y diseños.
- Se debe garantizar que los colores de interfaz de la aplicación sean claros. El color que predominará será el #00f06f para el banner y los botones de acción por defecto, mientras que el color principal será el #66666.

Portabilidad

- El sistema debe ser multiplataforma.

Usabilidad

- El software tendrá siempre visible una opción de ayuda, lo que posibilitará un mejor aprovechamiento por parte de los usuarios de sus funcionalidades.
- El sistema debe proporcionar mensajes de error que sean informativos y orientados al usuario final.

Especificaciones para el diseño y la implementación

- Emplear *JavaFX* para la interfaz visual.

Para establecer las capacidades operacionales que implementará la solución, se siguen los pasos que propone la metodología XP. A continuación se definen las Historias de Usuarios del sistema.

2.4 Fase I: Exploración

En esta etapa se confeccionan las Historias de Usuario (HU, en lo adelante) que describen las funcionalidades que el cliente desea estén presentes en la aplicación. El equipo de trabajo se familiariza con las herramientas que se emplearán en el desarrollo. Los programadores estiman los tiempos de desarrollo en base a la información ofrecida. Las estimaciones realizadas en esta fase son primarias, debido a que estarán basadas en datos de muy alto nivel y podrían variar cuando se analicen en cada iteración. Además, se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses (27).

2.4.1 Historias de Usuario

Las HU son la técnica utilizada para representar y especificar los requisitos del software. Se trata de un conjunto de tablas en las cuales el cliente describe de forma breve las características que el sistema debe poseer. El tratamiento de las HU es muy dinámico y flexible. Tienen el mismo propósito que los casos de uso de las metodologías tradicionales, las escriben los propios clientes tal y como ven ellos las necesidades del sistema. Las HU son descompuestas en tareas de programación y asignadas a los

programadores para ser implementadas durante las iteraciones que se definan (28). Están representadas mediante tablas con los siguientes elementos:

- **Número**: Número asignado a la HU.
- **Nombre de HU**: Atributo que contiene el nombre de la HU.
- **Fecha**: Fecha en la cual fue redactada la HU.
- **Usuario**: El usuario del sistema que utiliza o protagoniza la HU.
- **Prioridad en el negocio**: Contiene el nivel de prioridad de la HU en el negocio. Se clasifica en “Alta” en caso de que la HU sea indispensable en el negocio, “Media” en caso de que su realización no afecte el negocio y “Baja” cuando no se considera una prioridad pero resulta útil para el control de algunos elementos.
- **Riesgo de desarrollo**: Contiene el nivel de riesgo en caso de no realizarse la HU. Es considerado de “Alto” si el riesgo de no realizar la HU incide en el funcionamiento de la plataforma, “Medio” si el riesgo de no realizarla es medianamente importante y “Bajo” en caso de que no se considere un riesgo tardar en la realización de la HU y no incida en el funcionamiento de la plataforma.
- **Puntos estimados**: Este atributo es una estimación hecha por el equipo de desarrollo, aplicando la técnica de analogía, sobre el tiempo de duración de la HU. Cuando el valor es 1 equivale a una semana ideal de trabajo y un día equivale a 0.2 puntos.
- **Iteración asignada**: Especifica la iteración a la que pertenece la HU correspondiente.
- **Descripción**: Posee una breve descripción de lo que realizará la HU.
- **Observaciones**: Aquellos detalles relevantes que serán resueltos tras la conversación del equipo desarrollador con el cliente.

A continuación, se describen algunas HU correspondientes a la primera iteración de desarrollo definidas para llevar a cabo la implementación del sistema.

Tabla 1. HU1 Especificar un problema de estados.

Historia de Usuario

Número: 1	**Nombre de Historia de Usuario:** Especificar un problema de estados.
Usuario: **Usuario**	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Media	**Programador responsable:** Xiunellys Huerta Quintanilla
Descripción: Permite al usuario crear un problema de estados, al presionar el botón de crear se levanta una ventana para guardar el problema en un fichero con extensión **.pbm** y el nombre establecido.	
Observaciones: Una vez creado el problema se le mostrará al usuario la interfaz visual donde debe introducir la información del mismo para poder aplicar alguno de los algoritmos registrados en el sistema.	

Prototipo de Interfaz:

Tabla 2. HU5 Introducir estado.

Historia de Usuario	
Número: **3**	**Nombre de Historia de Usuario:** Introducir estado.
Usuario: **Usuario**	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0.2
Riesgo en Desarrollo: Media	**Programador responsable:** Xiunellys Huerta Quintanilla

Descripción:

Permite al usuario introducir un estado al problema. Los estados representan los nodos que conforman el espacio de búsqueda y pueden ser relacionados entre sí mediante una matriz de relaciones entre estados. Para ello se cuenta con un campo de entrada que adiciona los estados del problema. Campo:

- Estados

Observaciones: Una vez creado el estado, se genera una matriz de costo con la relación entre los estados que existan en el problema y un campo para indicar la heurística o la ganancia del mismo, esto resulta en dependencia del tipo de problema que sea especificado en el campo de selección “Tipo de problema” que puede tomar los valores: Camino mínimo, y Asignación.

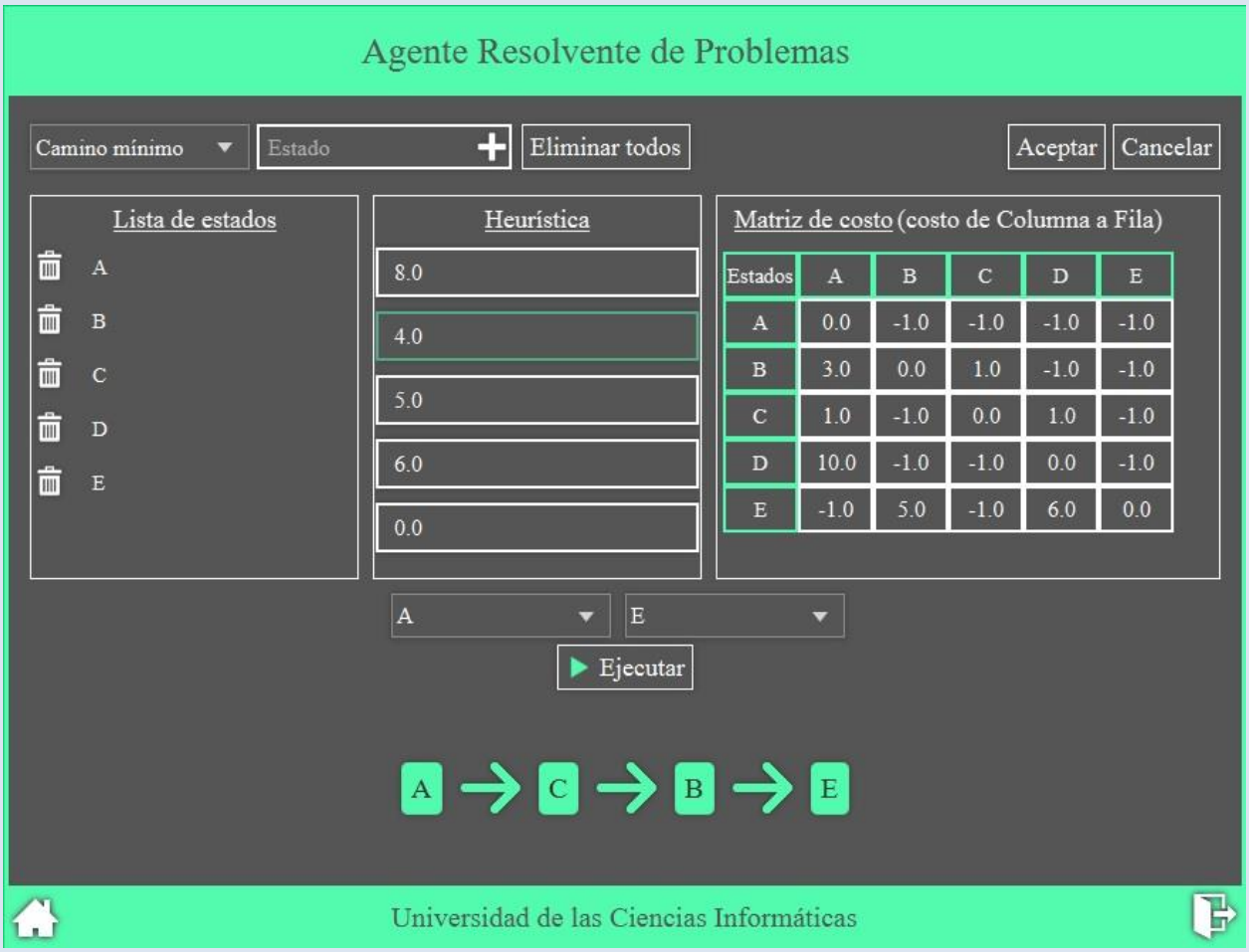
Prototipo de Interfaz:

Tabla 3. HU11 Aplicar algoritmo A.*

Historia de Usuario	
Número: **7**	**Nombre de Historia de Usuario*:*** Aplicar algoritmo A*.
Usuario: **Usuario**	**Iteración Asignada:** 1
Prioridad en Negocio: **Alta**	**Puntos Estimados:** 3

Riesgo en Desarrollo: **Alto**	**Programador responsable:** Xiunellys Huerta Quintanilla
Descripción: Permite al usuario aplicar el algoritmo A* sobre los estados de un problema al presionar el botón “Ejecutar”. Para ello es necesario especificar el tipo de problema y dependiendo de la selección, entonces se capturan los siguientes campos: Tipo de problema (Camino mínimo): • Estado inicial • Estado final • Heurísticas de los nodos que intervienen en el problema.	
Observaciones: Retorna el camino desde el estado inicial hasta el estado final atendiendo al conjunto de estados definidos en el problema.	

Prototipo de Interfaz:



2.5 Fase II: Planificación

En la fase de planificación se establece prioridad a las HU más importantes. Los desarrolladores estiman el esfuerzo necesario para cada HU y a partir de esto se define el plan de desarrollo de las iteraciones. Durante la primera iteración debe construirse la arquitectura del sistema (29).

2.5.1 Estimación del esfuerzo por HU

La estimación del esfuerzo necesario para el desarrollo de la aplicación mediante las HU consiste en la suma del tiempo establecido en cada una de las iteraciones donde se

distribuyeron esas HU. La siguiente tabla muestra las HU con los puntos de estimación en semanas.

Tabla 4. Estimación del esfuerzo por HU.

Historias de Usuario		Puntos estimados
1	Especificar un problema de estados.	1
2	Editar problema.	0.8
3	Introducir estado.	0.6
4	Modificar matriz de relaciones.	0.8
5	Introducir peso a un estado.	0.6
6	Especificar clasificación de un problema.	1.4
7	Aplicar algoritmo A*.	3
8	Aplicar algoritmo Escalador de colinas	3
9	Guardar problema.	0.4
10	Abrir problema existente.	0.6
11	Modificar estado.	0.4
12	Eliminar estado.	0.4
13	Eliminar todos los estados.	0.4
14	Mostrar ayuda del sistema.	0.2
15	Mostrar información del sistema.	0.2
Total		13.8

Una vez definidas las HU y realizada una previa estimación de esfuerzos, se procede a la planificación de la etapa de implementación del sistema. En este espacio, se crea el plan de iteraciones, donde se especifica la prioridad con que se implementarán las HU organizadas por iteraciones. Teniendo en cuenta el esfuerzo asociado a las HU y a las prioridades del cliente, se define una versión que sea de valor para este.

2.5.2 Plan de duración de las iteraciones

A continuación, se presenta el plan de duración de las iteraciones, donde cada iteración constituye la implementación de un grupo de HU que tributan a un resultado concreto de desarrollo. Este plan, tiene como finalidad, mostrar la duración de cada iteración, así como el orden en que serán implementadas las HU en cada iteración como se muestra en la tabla siguiente:

Tabla 5. Plan de duración de las iteraciones.

Iteración	Historias de Usuario		Puntos estimados
1	1	Especificar problema.	1
	2	Editar problema.	0.8
	3	Introducir estado.	0.6
	4	Modificar matriz de relaciones.	0.8
	5	Introducir peso a un estado.	0.6
	6	Especificar clasificación de un problema.	1.4
	7	Aplicar algoritmo A*.	3
2	8	Aplicar algoritmo Escalador de colinas	3
	9	Guardar problema.	0.4
	10	Abrir problema existente.	0.6

	11	Modificar estado.	0.4
	12	Eliminar estado.	0.4
	13	Eliminar todos los estados.	0.4
3	14	Mostrar ayuda del sistema.	0.2
	15	Mostrar información del sistema.	0.2
Total			13.8

2.5.3 Plan de entregas

En el plan de entrega que se plantea a continuación, se hace una propuesta de las versiones (*releases*) del sistema. Cada versión se conformará al finalizar una iteración.

Tabla 6. Plan de entregas del sistema.

Historia de Usuario	Final Iteración 1 (24 de marzo)	Final Iteración 2 (1 de mayo)	Final Iteración 3 (5 de mayo)
Especificar problema.	V1.0		
Editar problema.	V1.0		
Introducir estado.	V1.0		
Modificar matriz de relaciones.	V1.0		
Introducir peso a un estado.	V1.0		
Especificar clasificación de un problema.	V1.0		

Aplicar algoritmo A*.	V1.0		
Aplicar algoritmo *Escalador de Colinas*.		V1.1	
Guardar problema.		V1.1	
Abrir problema existente.		V1.1	
Modificar estado.		V1.1	
Eliminar estado.		V1.1	
Eliminar todos los estados.		V1.1	
Mostrar ayuda del sistema.			V1.2
Mostrar información del sistema.			V1.2

2.6 Fase III: Iteraciones

En esta fase se implementan las funcionalidades del sistema agente según la iteración que esté siendo desarrollada, se realizan las pruebas de aceptación para comprobar si se le dio cumplimiento a lo especificado en las HU. Si las pruebas fallan se procede a re-implementar las funcionalidades deficientes para solucionar el error. Cuando la aplicación de las pruebas es satisfactoria entonces se le entrega al cliente la versión recién implementada para que haga uso de las funcionalidades disponibles y se familiarice con la aplicación (28). En esta etapa el cliente se mantiene relacionado con el proceso de implementación para esclarecer posibles dudas en la comprensión del negocio.

2.7 Tarjetas CRC

Las HU son evaluadas para dividirlas en tareas, cada tarea representa una característica distinta del sistema y se puede diseñar una prueba de unidad que verifique cada tarea, estas tareas se representan por medio de las tarjetas CRC (Clase-Responsabilidad-Colaboradores). Cada tarjeta contiene el nombre de la clase, una descripción de las responsabilidades o métodos asociados con la clase, así como la lista de las clases con que se relaciona o que colaboran con ella (27). A continuación se describen las tarjetas definidas para la implementación de la solución.

Tabla 7. Tarjeta CRC MainWindowController.

Nombre de la clase: MainWindowController	
Responsabilidades	**Colaborador**
• handleBotonSalirAction(ActionEvent event). • handleBotonInicioAction(ActionEvent event).	• Agente. • ModelarProblemaController. • InicioController.

Tabla 8. Tarjeta CRC Agente.

Nombre de la clase: Agente	
Responsabilidades	**Colaborador**
• Percibir(ArrayList<Estado> listaDeEstados,float[][] matrizCostos) • Percibir(FileInputStream archivoParaAbrir)	• Percepcion. • Procesador. • Accion. • Problema

• Percibir(File archivoParaGuardar) • Percibir(Integer e_inicial, Integer e_final, Integer valor_Objetivo, HBox hbox_Resultado)	

Tabla 9. Tarjeta CRC ModelarProblemaController.

Nombre de la clase: ModelarProblemaController	
Responsabilidades	**Colaborador**
• handleBotonAddAction(ActionEvent event) • handleBotonElimTodoAction(ActionEvent event) • handleBotonAceptarAction(ActionEvent event) • handleBotonEjecutarAction(ActionEvent event) • handleBotonCancelarAction(ActionEvent event)	• *MainWindowController*. • Estado. • Problema.

Tabla 10. Tarjeta CRC Problema.

Nombre de la clase: Problema	
Responsabilidades	**Colaborador**
• Problema(ArrayList<Estado> estados, float[][] matriz_costo)	• Agente. • Estado.

• setEstados(ArrayList<Estado> estados) • setMatriz_costo(float[][] matriz_costo) • setTipo_problema(int Tipo_problema)	

Tabla 11. Tarjeta CRC A_Aster.

Nombre de la clase: A_Aster	
Responsabilidades	**Colaborador**
• Ejecutar(int e_inicial, int e_final) • generarSucesores() • tratar_Sucesor(Estado_A_Aster estado)	• Procesador. • Estado_A_Aster_CM. • Estado_A_Aster_Tab.

Tabla 12. Tarjeta CRC HillClimbing.

Nombre de la clase: HillClimbing	
Responsabilidades	**Colaborador**
• Ejecutar(Estado e_inicial, double objetivo) • evaluar_Estado (Estado estado, double objetivo) • generar_Estados(Estado mejor, double objetivo)	• Procesador. • Estado.

2.8 Conclusiones del capítulo

En este capítulo se han abordado los aspectos referentes a la concepción del producto a desarrollar y sus características funcionales. De esta forma podemos concluir que:

- A partir de la definición de las HU se pudieron identificar las principales funcionalidades a desarrollar en correspondencia con el sistema propuesto.
- La estimación del tiempo para la implementación de las HU definidas, permitió calcular una entrega final del producto en tres meses aproximadamente.
- Se exponen los artefactos generados en las fases de Exploración, Planificación y Diseño que establece la metodología XP.
- Con los aspectos arquitectónicos y de diseño a utilizar definidos, queda establecida la vía para llevar a cabo la implementación y prueba del sistema agente.

Capítulo 3: Implementación y pruebas del sistema

En este capítulo se analiza la propuesta de solución desde el punto de vista de la calidad, para evaluar cómo se le da cumplimiento al problema de la presente investigación y verificar si fueron implementadas de forma correcta cada una de las HU definidas por el cliente. Además, se definen las pautas para que el sistema sea implementado correctamente bajo estilos de programación. Se describen las tareas de ingeniería que conforman las HU y por último se llevan a cabo las pruebas de aceptación con el objetivo de comprobar que el sistema propuesto ha sido implementado de acuerdo a las funcionalidades especificadas.

3.1 Tareas de ingeniería

Las tareas de ingeniería pueden estar descritas por un lenguaje técnico y no ser necesariamente entendibles por el cliente. Tienen como objetivo definir cada una de las actividades que dan cumplimiento a las HU, de forma tal que se entienda lo que el sistema tiene que hacer y facilite su construcción (28). Se describen algunas de las tareas de ingeniería correspondientes a las HU del sistema.

- **HU Especificar problema**

Tabla 13. Tarea de ingeniería #1.

Tarea de ingeniería	
Número de tarea: 1	**Número de HU**: 1
Nombre de la tarea: Implementar ventana para localizar la ubicación donde guardar el problema que se desea crear.	
Tipo de tarea: Desarrollo	**Puntos estimados**: 0.4
Fecha de inicio: 23 de febrero de 2017	**Fecha fin**: 24 de febrero de 2017
Programador responsable: Xiunellys Huerta Quintanilla	

Descripción: Al presionar el botón “Crear un problema de estados”, se debe mostrar una ventana de navegación de acuerdo al explorador del sistema operativo donde se ejecute el agente. Esta ventana permite seleccionar una ubicación física en el disco duro de la PC y solo se debe necesitar especificar un nombre para guardar un fichero donde se almacenará la información relacionada con el problema que se esté trabajando en el sistema.

- **HU Introducir estado**

Tabla 14. Tarea de ingeniería #4.

Tarea de ingeniería	
Número de tarea: 4	**Número de HU**: 2, 3, 11, 12, 13
Nombre de la tarea: Actualizar componentes visuales de la interfaz de edición de problemas.	
Tipo de tarea: Desarrollo	**Puntos estimados**: 0.8
Fecha de inicio: 1 de febrero de 2017	**Fecha fin**: 3 de febrero de 2017
Programador responsable: Xiunellys Huerta Quintanilla	
Descripción: Implementar una función que permita actualizar los datos de cada componente visual que se encuentran en la interfaz de edición de problemas. Los principales componentes a actualizar son: • Componente para listar el nombre de los estados de un problema. • Componente para listar los pesos de los estados de un problema. • Componente para representar la matriz de relaciones entre los estados de un problema.	

Tabla 15. Tarea de ingeniería #5.

Tarea de ingeniería	
Número de tarea: 5	**Número de HU**: 3
Nombre de la tarea: Implementar componente visual para agregar estados a un problema.	
Tipo de tarea: Desarrollo	**Puntos estimados**: 0.4
Fecha de inicio: 6 de febrero de 2017	**Fecha fin**: 7 de febrero de 2017
Programador responsable: Xiunellys Huerta Quintanilla	
Descripción: Implementar un componente visual o campo de entrada que permita agregar estados al problema. Este componente debe permitir entrar nombre de estados formados solo por elementos alfanuméricos. Al crearse el elemento se debe poder visualizar el mismo en la lista de los estados del problema, en la lista de los pesos asociados a los nodos y en la matriz de relaciones entre nodos.	

- **HU Aplicar algoritmo A***

Tabla 16. Tarea de ingeniería #14.

Tarea de ingeniería	
Número de tarea: 14	**Número de HU**: 7
Nombre de la tarea: Implementar algoritmo A*.	
Tipo de tarea: Desarrollo	**Puntos estimados**: 2
Fecha de inicio: 6 de marzo de 2017	**Fecha fin**: 17 de marzo de 2017
Programador responsable: Xiunellys Huerta Quintanilla	

Descripción: Implementar una función que utilizando los datos del problema, genere el camino mínimo desde un estado inicial (o configuración inicial) a un estado final (o configuración final) especificado por el usuario, aplicando el algoritmo A*.

Tabla 17. Tarea de ingeniería #16.

Tarea de ingeniería	
Número de tarea: 16	**Número de HU**: 7
Nombre de la tarea: Mostrar camino hallado.	
Tipo de tarea: Desarrollo	**Puntos estimados**: 0.5
Fecha de inicio: 22 de marzo de 2017	**Fecha fin**: 24 de marzo de 2017
Programador responsable: Xiunellys Huerta Quintanilla	
Descripción: Implementar y diseñar un componente visual en la interfaz de edición de problemas, que permita mostrar el camino encontrado mediante el algoritmo A*.	

- **HU Aplicar algoritmo *Escalador de colinas***

Tabla 18. Tarea de ingeniería #17.

Tarea de ingeniería	
Número de tarea: 17	**Número de HU**: 8
Nombre de la tarea: Implementar algoritmo *Escalador de colinas.*	
Tipo de tarea: Desarrollo	**Puntos estimados**: 2
Fecha de inicio: 27 de marzo de 2017	**Fecha fin**: 7 de abril de 2017

Programador responsable: Xiunellys Huerta Quintanilla
Descripción: Implementar una función que, utilizando los datos del problema, genere el estado final al que se desea llegar, aplicando el algoritmo *Escalador de colinas*.

Tabla 19. Tarea de ingeniería #19.

Tarea de ingeniería	
Número de tarea: 19	**Número de HU**: 8
Nombre de la tarea: Mostrar estado final.	
Tipo de tarea: Desarrollo	**Puntos estimados**: 0.5
Fecha de inicio: 12 de mayo de 2017	**Fecha fin**: 14 de mayo de 2017
Programador responsable: Xiunellys Huerta Quintanilla	
Descripción: Implementar y diseñar un componente visual en la interfaz de edición de problemas, que permita mostrar el estado encontrado mediante el algoritmo *Escalador de colinas*.	

Para llevar a cabo la implementación de las tareas se hace necesario definir un grupo de estilos de programación que guiarán la forma en que se escribirá el código de la aplicación a desarrollar.

3.2 Estilos de programación

Un estilo de programación es un término que describe convenciones para escribir código fuente en ciertos lenguajes de programación, además de brindarle una mayor legibilidad al código. En la implementación de la aplicación se utilizaron los estándares de programación que define el propio lenguaje Java, así como algunos estilos que se describen a continuación.

3.2.1 Definición de clases

Los nombres de las clases comenzarán con letra inicial mayúscula y en caso de ser nombres compuestos, cada una comenzará con letra inicial mayúscula y unidos por un guion bajo. Además, la llave de apertura se colocará a continuación de la declaración de la clase. Seguidamente se muestra un ejemplo de declaración de clase.

```
public class Estado_A_Aster extends Estado {
    private Estado_A_Aster anterior;
    private List<Estado_A_Aster> sucesores;
    private float costo;
```

Ilustración 9. Definición de clases.

3.2.2 Definición de métodos

El nombre de los métodos comienza con una letra minúscula, en caso de ser un nombre compuesto por dos o más palabras, estas comenzarán con letra mayúscula. A continuación un ejemplo:

```
public ArrayList<Estado> getEstados() {
    return estados;
}

public void setEstados(ArrayList<Estado> estados) {
    this.estados = estados;
}
```

Ilustración 10. Definición de métodos.

3.2.3 Llamadas a funciones y asignación de variables

Las llamadas a las funciones de una clase se realizarán utilizando un punto entre el nombre de la clase y el del método, sin espacios entre el nombre de la función y los paréntesis. En el caso de utilizar llamadas a funciones de una misma clase no se utilizará ningún precedente. Las asignaciones se realizarán mediante el signo de igualdad (=). A continuación se muestra un ejemplo:

```
mejorEstado = estados_Abiertos.poll();
lista_estados_Abiertos.remove(mejorEstado);
estados_Cerrados.add(mejorEstado);
lista_estados_Cerrados.add(mejorEstado);
```

Ilustración 11. Llamadas a funciones.

3.2.4 Estructuras de control

Con el objetivo de mejorar la legibilidad del código fuente se define que las estructuras de control deben estar identadas. Las estructuras de control incluyen ***if***, ***for***, ***while***, entre otras como se aprecia a continuación:

```
if (estados_Abiertos.isEmpty()) {
    return false;
} else {
    mejorEstado = estados_Abiertos.poll();
    lista_estados_Abiertos.remove(mejorEstado);
    estados_Cerrados.add(mejorEstado);
    lista_estados_Cerrados.add(mejorEstado);
    if (mejorEstado.equals(Agente.problema.getEstados().get(e_final))) {
        soluc_encontrada = true;
        ((Estado_A_Aster) Agente.problema.getEstados().get(e_final)).setAnterior(mejorEstado.getAnterior());
    } else {
        generarSucesores();
        List<Estado_A_Aster> sucesores = mejorEstado.getSucesores();
        for (int i = 0; i < sucesores.size(); i++) {
            tratar_Sucesor(sucesores.get(i));
            System.out.println(sucesores.get(i).getNombre() + "  " + sucesores.get(i).getCosto());
        }
    }
```

Ilustración 12. Estructuras de control.

Una vez definida la forma de codificar se realiza la implementación de cada HU definida según la planificación realizada, pero es posible que se hayan cometido algunos errores de implementación a lo largo del proyecto, por lo que se hace necesario comprobar que la herramienta funcione correctamente. Con el objetivo de comprobar la correcta implementación de las funcionalidades del sistema, a continuación se describen las principales pruebas realizadas, así como los resultados obtenidos.

3.3 Pruebas del sistema

Las pruebas son consideradas una de las fases más importantes en todo proyecto que se lleve a cabo, ya que estas permiten corregir la mayor cantidad de errores posibles y entregar un producto de mejor calidad. Para la validación funcional del presente producto se realizaron dos tipos de pruebas: unitarias, elaboradas por el desarrollador para evaluar automáticamente el desempeño individual de las funcionalidades; y las pruebas de aceptación, que se elaboran en relación con las HU para validar que el sistema cumpla con las especificaciones realizadas por el cliente. Para la ejecución de las pruebas unitarias se emplea el *framework JUnit*, un paquete de pruebas que permite realizar este tipo de acciones en Java y puede ser integrado fácilmente con el IDE *NetBeans*. Mientras que para las pruebas de aceptación se realizó la técnica de partición de equivalencia.

3.3.1 Pruebas unitarias

Es un método de prueba de software que se realiza sobre las funciones internas de un módulo. Se llevan a cabo, en primer lugar, sobre un módulo concreto, para luego realizar las pruebas unitarias utilizando el método de caja blanca sobre varios subsistemas (integración) (19). Están centradas en el código y le permiten al desarrollador comprobar si las funciones o algoritmos implementados en el software tienen la calidad requerida o si tienen algún error. Puede proveer la solución a cualquier error que se encuentre en el código a la hora de probar, o reportar a los desarrolladores la solución al desperfecto detectado y no solo la existencia del mismo.

Se recomienda que estas pruebas sean automatizadas, pues esto facilita al programador identificar funciones que no ofrecen una salida acorde con la lógica que se deseaba implementar. La herramienta *JUnit* permite correr un conjunto de pruebas de forma automática e informa de aquellas que han fallado.

```
public static boolean Ejecutar(int e_inicial, int e_final) {
    estados_Abiertos_CM = new PriorityQueue<>();
    lista_estados_Abiertos_CM = new ArrayList<>();
    lista_estados_Cerrados_CM = new ArrayList<>();

    Estado_A_Aster_CM inicial = (Estado_A_Aster_CM) Agente.problema.getEstados().get(e_inicial);
    inicial.setCosto(0);
    estados_Abiertos_CM.add(inicial);
    lista_estados_Abiertos_CM.add(inicial);

    boolean soluc_encontrada = false;
    do {
        if (estados_Abiertos_CM.isEmpty()) {
            return false;
        } else {
            mejorEstado_CM = estados_Abiertos_CM.poll();
            lista_estados_Abiertos_CM.remove(mejorEstado_CM);
            lista_estados_Cerrados_CM.add(mejorEstado_CM);
            if (mejorEstado_CM.equals(Agente.problema.getEstados().get(e_final))) {
                soluc_encontrada = true;
                ((Estado_A_Aster_CM) Agente.problema.getEstados().get(e_final)).setAnterior(mejorEstado_CM.getAnterior());
            } else {
                generarSucesores();
                List<Estado_A_Aster_CM> sucesores = mejorEstado_CM.getSucesores();
                for (int i = 0; i < sucesores.size(); i++) {
                    tratar_Sucesor(sucesores.get(i));
                }
            }
        }
    } while (!soluc_encontrada);

    return true;
}
```

Ilustración 13. Código del método Ejecutar de la clase A_Aster.

```
@Test
public void testEjecutar_int_int() {
    System.out.println("Ejecutar");
    int e_inicial = 1;
    int e_final = 5;
    boolean expResult = true;
    boolean result = A_Aster.Ejecutar(e_inicial, e_final);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}
```

Ilustración 14. Caso de prueba unitaria del método Ejecutar de la clase A_Aster.

Los casos de pruebas fallidos fueron corregidos antes del cierre de la iteración en que se evaluaba. Se tomaron precauciones para evitar errores similares y se definieron nuevos casos de prueba para verificar el correcto desempeño de las funcionalidades. En total se implementaron 8 casos de pruebas unitarias distribuidas de la siguiente manera:

4 pruebas para ejecutar el algoritmo A* sobre problemas de camino mínimo, mientras que se diseñaron 4 pruebas para comprobar la ejecución del algoritmo Escalador de Colinas sobre problemas de asignación. De todas ellas 3 aportaron no conformidades y se corrigieron antes de terminar la iteración. La mayoría de los casos de prueba están relacionados con las funciones del agente inteligente ya que su correcto funcionamiento es vital en el desarrollo de la aplicación, los resultados aparecen en el siguiente gráfico.

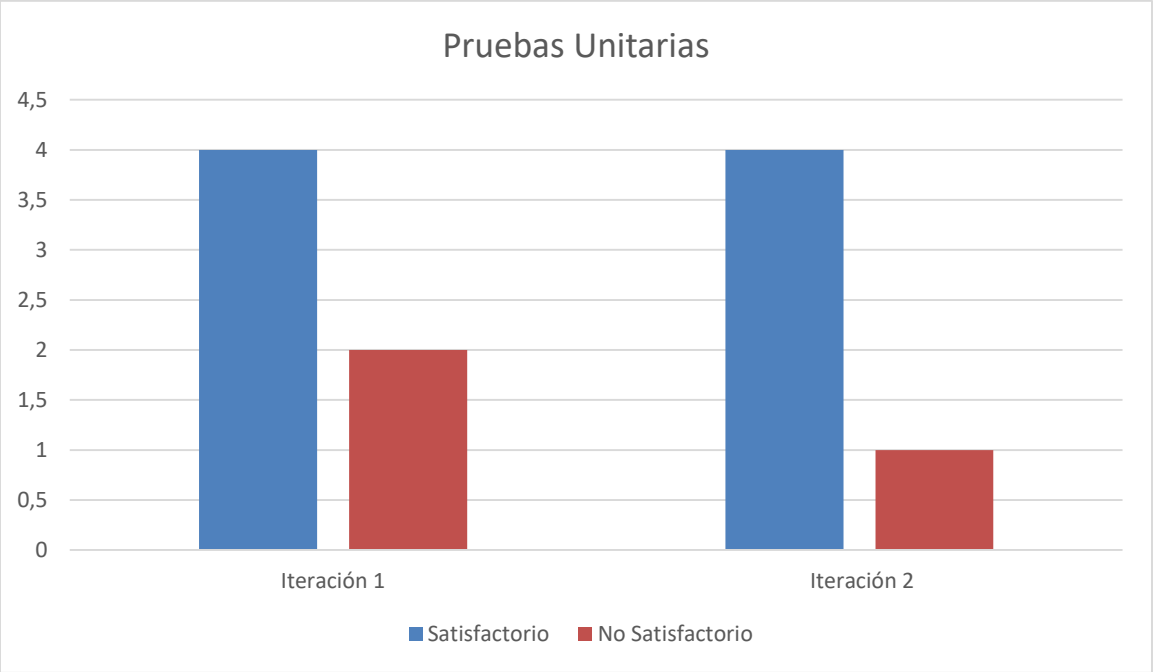


Ilustración 15. Resultados de la aplicación de los casos de pruebas unitarias.

3.3.2 Pruebas de aceptación

Las pruebas de aceptación son especificadas por el cliente, se centran en las características y funcionalidades generales del sistema que son visibles. Estas pruebas derivan de las HU que se han implementado como parte de la liberación del software. Una prueba de aceptación es con el método de caja negra. Cada una de ellas representa una salida esperada del sistema (30). Es responsabilidad del cliente verificar la corrección y toma de decisiones acerca de estas pruebas.

En la confección de estas pruebas se describen los siguientes aspectos:

- **Código**: Identifica la prueba en cuestión, incluye el número de HU, de la prueba y si posee diferentes escenarios.
- **Historia de Usuario**: Número de la HU a la cual pertenece.
- **Nombre**: Nombre del caso de prueba.
- **Descripción**: Acciones que debe realizar el sistema.
- **Condiciones de ejecución**: Describe las características y elementos en general que debe tener el sistema para realizar el caso de prueba.
- **Pasos de Ejecución**: Incluye las entradas necesarias del sistema y los pasos necesarios para realizar el caso de prueba.
- **Resultados**: Clasificación de la prueba en satisfactoria o insatisfactoria.

A continuación, se muestra una representación de las pruebas de aceptación a realizarse en las diferentes iteraciones de desarrollo.

Pruebas de aceptación para la iteración 1

Para la primera iteración, se definieron un total de 14 casos de pruebas de aceptación. Todas enfocadas a evaluar la implementación del manejo de los problemas de estados. Se describen algunas de las pruebas realizadas, el resto pueden ser consultadas en los anexos de la investigación.

Tabla 20. Prueba de aceptación #1.

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de Usuario**: 1
Nombre: Crear un problema de estados.	
Descripción: Se crea un problema de estados, poniendo como nombre “prueba_aceptacion1”.	
Condiciones de ejecución:-	
Pasos de ejecución:	

1. Acceder a la vista principal del sistema. 2. Presionar el botón crear problema. 3. Seleccionar la ubicación donde guardar el problema. 4. Especificar como nombre “prueba_aceptacion1”. 5. Presionar el botón “Guardar”.
Resultado: Satisfactorio

Tabla 21. Prueba de aceptación #2.

Caso de prueba de aceptación	
Código: HU1_P2	**Historia de Usuario**: 1
Nombre: Abandonar ventana para guardar un nuevo problema.	
Descripción: Se intenta crear un nuevo problema de estados y se abandona la acción.	
Condiciones de ejecución:-	
Pasos de ejecución: 1. Acceder a la vista principal del sistema. 2. Presionar el botón crear problema. 3. Seleccionar la ubicación donde guardar el problema. 4. Presionar el botón “Cancelar”.	
Resultado: Satisfactorio	

Tabla 22. Prueba de aceptación #3.

Caso de prueba de aceptación	
Código: HU3_P1	**Historia de Usuario**: 3

Nombre: Introducir un estado a un problema previamente creado.
Descripción: Se introduce un estado denominado “estado_prueba1”.
Condiciones de ejecución: Debe existir un problema de estados abierto.
Pasos de ejecución: 1. Localizar componente visual para introducir un nuevo estado al problema actual. 2. Escribir el nombre del nuevo estado “estado_prueba1”. 3. Presionar el botón “Adicionar estado”.
Resultado: Satisfactorio

Tabla 23. Prueba de aceptación #4.

Caso de prueba de aceptación	
Código: HU7_P1	**Historia de Usuario**: 7
Nombre: Ejecutar algoritmo *A** sobre un problema de estados.	
Descripción: Se ejecuta el algoritmo *A** sobre un problema de estados determinado.	
Condiciones de ejecución: Debe existir un problema de estados abierto y este debe contener al menos un estado.	
Pasos de ejecución: 1. Indicar el tipo de problema como “Camino mínimo”. 2. Indicar estado inicial. 3. Indicar estado final. 4. Presionar el botón Ejecutar. 5. Aceptar la ejecución del algoritmo seleccionado.	

Resultado: Satisfactorio

Se desarrollaron un total de 24 casos de pruebas de aceptación. Estas pruebas fueron realizadas de forma organizada, por cada iteración definida. A continuación, se muestra en gráficas, los porcientos de satisfacción alcanzados en cada iteración.

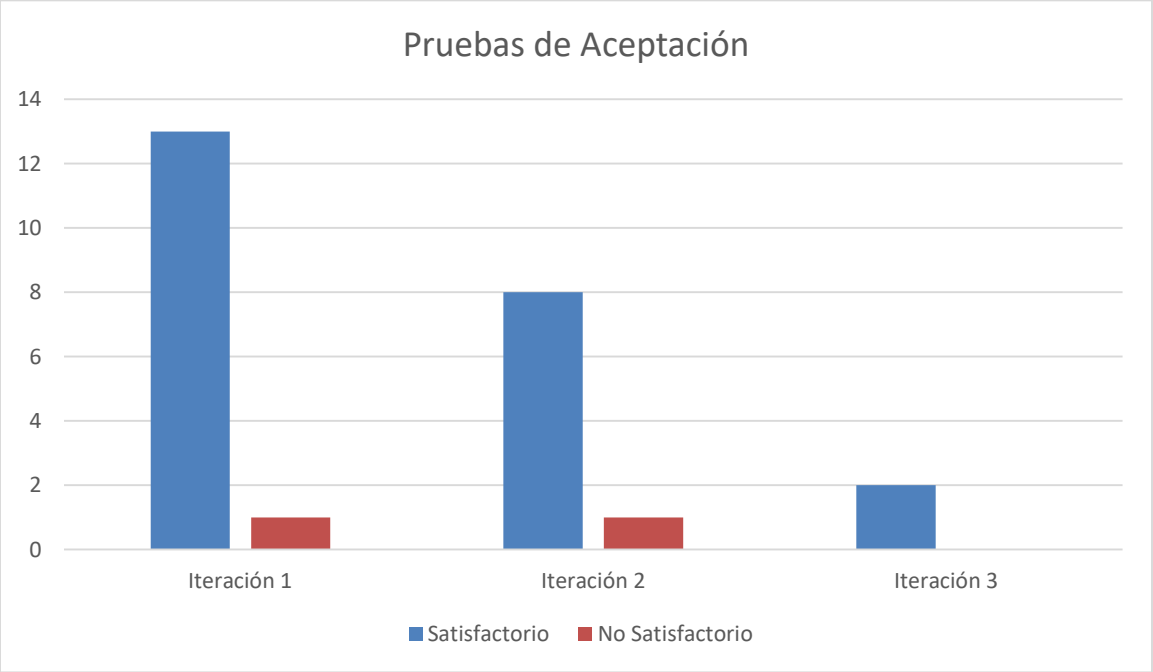


Ilustración 16. Resultados de las pruebas de aceptación.

Como se puede observar en la representación, en la primera iteración se realizaron un total de 14 pruebas, de ellas 13 alcanzaron el nivel de satisfacción esperado para un 92.86%. En tanto, en las iteraciones II y III, se realizaron 8 y 2 pruebas respectivamente, donde todas son evaluadas de resultado satisfactorio excepto una en la segunda iteración, la cual detectó que se guardaba la matriz de relaciones entre estados de forma inversa y al cargar los valores del problema la matriz quedaba invertida. Esta acción fue corregida con la implementación adecuada. Con estas comprobaciones, se obtiene un 100% de satisfacción en la iteración final del producto, lo cual corrobora el correcto funcionamiento de las funcionalidades implementadas.

Conclusiones

Con el desarrollo de la presente investigación se arriba a las siguientes conclusiones:

- La metodología XP se ajusta a las necesidades del desarrollo del sistema propuesto, por lo que fue seleccionada para guiar el proceso de desarrollo del agente basado en el modelo BDI.
- Se obtiene un agente inteligente capaz de resolver problemas de búsqueda heurística aplicando los algoritmos Escalador de Colinas y A* en correspondencia con problemas de tipo Asignación y Camino Mínimo respectivamente.
- Tanto las pruebas unitarias como las pruebas de aceptación reflejaron resultados favorables, con lo cual se considera cumplido el objetivo trazado en la investigación.

Recomendaciones

A partir de los resultados obtenidos se recomienda:

Incorporar la implementación de algoritmos meta heurísticos para aumentar el campo de acción del agente.

Referencias Bibliográficas

1. *Métodos de solución de problemas de la IA.* **Pérez, Rafael Bello.** Santa Clara : Grupo de Investigaciones en Inteligencia Artificial. Departamento de Ciencia de la Computación. Universidad Central de Las Villas. CUBA, 1998.

2. **Malagon, C.** Docplayer.es. *Búsqueda heurística Prof. Constantino Malagón.* [En línea] 2016. [Citado el: 20 de 1 de 2017.] http://docplayer.es/10721791-Busqueda-heuristica-prof-constantino-malagon.html.

3. **Russell, Stuart and Norving, Petter.** *Artificial Intelligence. A Modern Approach.* s.l. : PEARSON Prentice Hall, 2010.

4. **López, B.** Hill Climbing. *Hill Climbing.* s.l. : Instituto Tecnológico de Nuevo Laredo, 2005.

5. **Jones, M. Tim.** *Artificial Intelligence: A Systems Approach: A Systems Approach. Jones & Bartlett Learning.* 2015.

6. *Comparison of simulated annealing and hill climbing in the course timetabling problem.* **Patrick, Kenekayoro.** s.l. : African Journal of Mathematics and Computer Science Research, 2012, Vol. 5.

7. *Un nuevo algoritmo para la optimización de estructuras: El Recocido Simulado.* **Espín, Mariano Vázquez.** Madrid : s.n., 2012.

8. **Facultad Informática de Barcelona.** *Apuntes de Inteligencia Artificial.* Barcelona : s.n., 2013.

9. *La red semántica.* **Berners-Lee, Tim and Hendler, James and LASILA, Ora.** s.l. : Investigación y Ciencia, 2001.

10. **TIOBE.** Home | TIOBE - The Software Quality Company. *Tiobe.com.* [En línea] 2017. [Citado el: 23 de 1 de 2017.] http://www.tiobe.com/.

11. **JAVA.** Java.com. *¿Qué es Java y para qué es necesario?* [En línea] 2017. [Citado el: 23 de 1 de 2017.] https://www.java.com/es/download/faq/whatis_java.xml.

12. *Sistema inteligente para la recomendación de objetos de aprendizaje.* **Casali, Ana, y otros.** 2011.

13. *Un sistema de tutoría inteligente adaptativo considerando estilos de aprendizaje.* **Peña, Clara Inés, y otros.** 2, s.l. : Revista UIS ingenierías, 2012, Vol. 1.

14. *Sistema de vigilancia de personas mayores o con incapacidad usando sistemas multiagente y robots bípedos controlados mediante voz.* **Arce, Alex Yanahuaya.** 2016.

15. **NETBEANS.** Netbeans.org. *Netbeans.org.* [En línea] 2017. [Citado el: 23 de 1 de 2017.] https://netbeans.org/.

16. **ECLIPSE.** Eclipse.org. *Eclipse.org.* [En línea] 2017. [Citado el: 24 de 1 de 2017.] https://eclipse.org/ide/.

17. **JETBRAINS.** JetBrains.com. *JetBrains.com.* [En línea] 2017. [Citado el: 24 de 1 de 2017.] https://www.jetbrains.com.

18. **ORACLE.** Oracle.com. *JavaFX Developer Home.* [En línea] 2017. [Citado el: 24 de 1 de 2017.] http://www.oracle.com/technetwork/java/javafx/overview/index.html.

19. **Schenone, Hernán Marcelo.** *Diseño de una Metodología Ágil de Desarrollo de Software.* 2004.

20. **Pressman, R. S.** *Ingeniería del Software: una tecnología estratificada. Ingeniería del Software. Un enfoque práctico.* 2003.

21. **Schwaber, K.** *Scrum development process. Business Object Design and Implementation.* 1997.

22. **Suterland, Ph. D.** *Distributed Scrum: Agile project management with outsourced development teams.* 2007.

23. *Elaboración del TFG en Ingeniería Informática en Sistemas de Información a partir de metodologías ágiles.* **Gallego, Román y Ángel, Jesús.** 2016.

24. **NÚÑEZ MORI, JOSE GERMÁN .** *USABILIDAD EN METODOLOGÍAS ÁGILES.* 2010.

25. **Knublauch, Holger.** Extreme programming of multi-agent systems. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2.* 2002.

26. *UML 2.0 and agents: how to build agent-based systems with the new UML standard.* **Bauer, Bernard y Odell, James.** s.l. : Engineering applications of artificial intelligence, 2005, Vol. 18.

27. **Longo, Francesco y Iazzolino, Stefano.** Agile Software Development: A Modeling and Simulation Showcase in Military Logistics. *Proceedings of 4th International Conference in Software Engineering for Defence Applications.* 2016.

28. *Sistema multi-agente deliberativo para la obtención y análisis de datos en Honeynets.* **Giraldo, EM and Isaza, GA.** 20, s.l. : Entre Ciencia e Ingeniería, 2016.

29. **Joskowicz, Ing. José.** *Reglas y Prácticas en eXtreme Programming.* España : s.n., 2008.

30. **Letelier, Patricio y Penades, Mª Carmen.** *Métodologías ágiles para el desarrollo de software:eXtreme . Programming (XP).* 2013.

31. **Escribano, Gerardo Fernández.** *Introducción a Extreme Programming.* 2002.

32. *Algoritmos de búsqueda heurística en tiempo real. Aplicación a la navegación en los juegos de video.* **Fernández, M.** 2005.

33. **Larman, Craig.** *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* s.l. : Pearson, 2002.