UNIVERSIDAD DON BOSCO
FACULTAD DE INGENIERIA
ESCUELA DE INGENIERIA EN COMPUTACION

PLANIFICACIÓN DE TRAYECTORIAS DE OBJETOS MÓVILES EN EL PLANO Y SIMULACION TRIDIMENSIONAL DE LAS MISMAS

PROYECTO DE GRADUACION
PARA OPTAR AL GRADO DE
INGENIERO EN COMPUTACIÓN

PRESENTADO POR:
ANA LUISA ALAM ZAMORA
JUAN CARLOS ZÁRATE DERAS

CIUDADELA DON BOSCO Mayo 2003

ÍNDICE

ÍNDICE DE IMÁGENES

- **Figuras**

- **Tablas**

INTRODUCCIÓN

Muchos problemas de la vida real se resuelven por medio de algoritmos de búsqueda. Tal es el caso de aplicaciones para administración de redes, determinación de trayectorias, robot móvil, secuencia de un ensamble, etc. La planificación de trayectorias para robots no es la excepción. Este trabajo comprende la planificación de trayectorias para objetos móviles, que a diferencia de los robots, no heredan sus propiedades físicas y eléctricas.

Se ha utilizado el algoritmo de búsqueda A*, tomado como referencia en temas de búsqueda en el campo de Inteligencia Artificial, para resolver el problema de planificación de trayectorias para objetos móviles. Con el objetivo de verificar como se puede llevar a la práctica dicho algoritmo, se realiza una simulación de la navegación del móvil a partir de la trayectoria resuelta por el algoritmo. Esta simulación es realizada en dos y tres dimensiones.

Con este trabajo pretendemos abarcar y describir los conceptos básicos de los elementos que forman parte de la aplicación, los cuales se enumeran a continuación: el método de descomposición en celdas para planificación de trayectorias, el algoritmo de búsqueda A* y la graficación en tercera dimensión. Además se detallan las estructuras, módulos, clases y funciones que fueron utilizadas y escritas en el lenguaje de programación Visual Basic.NET para crear la interfaz de la aplicación y codificación del algoritmo A*.

Por último se describe el procedimiento utilizado para conectar la interfaz de Visual Basic.NET con las librerías de OpenGL para la simulación en tercera dimensión de la navegación del móvil.

1. Planteamiento del trabajo de Graduación

1.1 ANTECEDENTES

La Inteligencia Artificial es una de las disciplinas más recientes. El término fue dado por McCarthy en el año 1956 durante un taller desarrollado en Darthmouth College que pretendía reunir a investigadores estadounidenses interesados en la teoría de autómatas, redes neuronales y el estudio de la inteligencia. Desde entonces, sus diferentes ramas han sido objeto de estudio e investigación por parte de instituciones de educación de nivel superior y organismos dedicados al desarrollo de la ciencia y la tecnología.

Una definición aceptada de Inteligencia Artificial fue dada por SchalKoffen en 1990: “Un campo de estudio que se enfoca a la explicación y emulación de la conducta inteligente en función de procesos computacionales”.

Durante los últimos años, las investigaciones sobre Inteligencia Artificial han sido orientadas a retomar teorías existentes y demostrar su utilidad en la resolución de problemas reales de ingeniería, crear nuevas oportunidades de negocios y en muchas otras áreas de aplicación.

Los esfuerzos en nuestro país por estudiar, investigar y desarrollar esta disciplina han sido escasos y hasta cierto punto, nulos. Apenas unos cuantos trabajos realizados en este campo pueden ser encontrados en las bibliotecas de nuestras universidades y asignaturas formales que traten sobre Inteligencia Artificial no son incluidas en de los planes de estudio universitarios.

La elaboración de un trabajo que utilice algoritmos de Inteligencia Artificial se convierte, en este contexto, en precedente y material útil para aquellas generaciones que deseen generar soluciones a problemáticas reales e introducirse en este campo. Este trabajo pretende convertirse en una motivación para que se inicie una incursión de la población estudiantil en materias de alto nivel tecnológico y de gran utilidad como lo es la Inteligencia Artificial.

1.2 IMPORTANCIA Y JUSTIFICACIÓN DEL TEMA

La detección de colisiones es un área de gran importancia para materias tales como la Graficación y Simulación en Realidad Virtual. La planificación de movimientos libres de colisiones se convierte en un reto, en el momento en que uno o más objetos se mueven en un medio donde existen obstáculos.

Para poder realizar la planificación de los movimientos de un objeto dentro del espacio de trabajo, es necesario realizar una representación de dicho espacio y del objeto mismo. Una vez representada la escena se procede a hacer la planificación de rutas, verificando que no exista colisión con alguno de los obstáculos del medio

El proyecto busca las particularidades del problema de la planificación de trayectorias que puedan ser utilizadas para optimizar el cálculo de la ruta. Con la finalidad de optimizar la trayectoria del objeto móvil, de un gran número de algoritmos utilizados en Inteligencia Artificial para resolver problemas de búsqueda sobre rutas o grafos, se utilizará el algoritmo A*, que cumple con la característica de encontrar rutas óptimas.

Cabe señalar que la inteligencia artificial no es más que uno de los muchos campos en los que la búsqueda es un tema importante, y que muchos algoritmos de búsqueda son utilizados en aplicaciones tales como el ruteo de las redes de cómputo, sistemas para la planificación de los viajes aéreos, etc.

Actualmente en el país, son muy pocas las aplicaciones que involucran conceptos de inteligencia artificial y gráficos por computador que demuestren su utilidad y, que permitan por un lado, generar mejores soluciones a problemas en el campo tecnológico y por otro, que sirvan de base para el desarrollo de proyectos de este tipo.

1.3 PROYECCIÓN SOCIAL

Esta investigación pretende ofrecer resultados teóricos relacionados con la planificación de trayectorias para objetos móviles autónomos y ejemplificar dichos resultados mediante una simulación.

La finalidad de desarrollar una aplicación como la tratada en este trabajo, es fijar un precedente a la comunidad estudiantil de las universidades salvadoreñas y brindar un documento de consulta a toda aquella persona interesada en introducirse en los campos de Inteligencia Artificial y gráficas en tercera dimensión

Los sectores objetivo de este trabajo debemos identificarlos en función de lo que queremos para nuestro país; por lo tanto es importante definir con claridad nuestra visión a largo plazo.

Consideramos que El Salvador debe de iniciar un proceso de desarrollo sostenible y sustentable en el tiempo, que permita el desarrollo económico y social de nuestro país y la sociedad salvadoreña pueda mejorar su calidad de vida. Este desarrollo debe de tener como fundamento la generación de riqueza, partiendo del conocimiento y el desarrollo tecnológico y no de la especulación.

Para ello es importante fomentar el desarrollo tecnológico en nuestras universidades, formando generaciones de profesionales con visión estratégica de futuro, para los cuales, los logros de tecnologías de avanzada de otros países no sean solo materiales de lectura, sino que por el contrario, se transformen en retos

para su creatividad y crecimiento personal, desarrollando por ellos mismos nuevas tecnologías que les permita competir en el mercado profesional, la comunidad de estudiantes universitarios de la carrera de computación y es definitivamente uno de los sectores objetivo de esta investigación.

Si el mercado laboral salvadoreño ofrece profesionales altamente calificados en la materia objeto de este estudio, resulta atractivo para los sectores económicos invertir en proyectos relacionados, promoviendo con ello un factor de desarrollo. Despertar al sector inversionista sobre la posibilidad de hacer buenos negocios, con mano de obra nacional altamente calificada, es otro de los sectores objetivo que nos proponemos.

Finalmente, aunque resulte pretencioso, es importante demostrar al sector gubernamental las potencialidades de nuestras universidades en fomentar las capacidades de crear y desarrollar tecnologías, y con ello transformarse en verdadero dinamizador de la investigación científica y tecnológica, debido a ello, el sector gubernamental es nuestro tercer objetivo.

1.4 DEFINICIÓN DEL TEMA

El tema del presente trabajo lo podemos formular de la siguiente manera:

"Planificación de trayectorias de objetos móviles en el plano
y simulación tridimensional de las mismas"

Si entendemos por planificación al "análisis por adelantado de una secuencia entera de pasos para descubrir a dónde se conducirá antes de dar el primer paso" (Rich y Knight,1994:52) y por trayectoria a "la línea descrita en el espacio por un punto que se mueve" (Diccionario de la Lengua Española, 1992) se puede concebir el grado de trabajo necesario para realizar el movimiento de un objeto móvil alrededor del mundo evitando obstáculos durante su desplazamiento.

Para la comprensión del desplazamiento del objeto se contempla la colocación de éste en un ambiente en tercera dimensión, de tal forma que se visualice el recorrido del objeto a través de la trayectoria previamente trazada.

1.5 OBJETIVOS

1.5.1 Objetivos Generales

Los objetivos generales que se persiguen con el desarrollo de este trabajo son:

- Utilizar técnicas y conceptos de Inteligencia Artificial en la planificación de una trayectoria libre de obstáculos que conduzca a un objeto móvil desde una posición inicial hasta una de destino.

- Sentar las bases para que futuros trabajos de graduación retomen el tema y se desarrollen nuevas metodologías de planificación de trayectorias para objetos móviles.

1.5.2 Objetivos Específicos

Los objetivos específicos pueden enunciarse de la siguiente manera:

- Presentar los conceptos involucrados para la resolución de problemas en la navegación de objetos.

- Aplicar el algoritmo de búsqueda A* en la planificación de trayectorias de objetos móviles

- Simular en tercera dimensión la trayectoria y el desplazamiento de un objeto móvil utilizando OPENGL.

1.6 ALCANCES Y LIMITACIONES

1.6.1 ALCANCES

- Documentar los conceptos básicos de graficación en tercera dimensión.

- El usuario podrá configurar la disposición de los obstáculos en el área de trabajo donde se desplazará el objeto.

- La aplicación brindará al usuario la opción de colocar ciertos obstáculos, elegibles de una lista, dentro del ambiente en el que se ubicará al móvil. Los obstáculos de dicha lista cumplirán con las siguientes características:
    - Estáticos y rígidos. Los obstáculos no se desplazarán durante la simulación.
    - Su forma y tamaño serán definidos por el usuario a partir de una lista limitada.

- El usuario elegirá las posiciones de inicio y destino de la trayectoria del móvil

- Se utilizará el algoritmo A* y el método de descomposición en celdas para resolver el problema de planificación de trayectorias de un objeto móvil.

- El objeto móvil será un cuerpo rígido cuya forma será elegida por el usuario de una lista predefinida y cuyas dimensiones serán indicadas a partir de una escala continua.

- La trayectoria y el desplazamiento del móvil será simulada en dos y tres dimensiones.

- Se proveerán al menos dos vistas en tercera dimensión de la simulación

- La aplicación poseerá la opción de representar en dos y tres dimensiones la solución generada por otros algoritmos implementados en diferentes lenguajes de programación, utilizando como conexión un archivo de texto entre las aplicaciones.

- La aplicación presentará opciones de almacenar la configuración del ambiente gráfico en dos dimensiones como una imagen del tipo bitmap y los resultados obtenidos por la ejecución del algoritmo para futuros análisis.

1.6.2 LIMITACIONES

Las limitaciones que presenta la aplicación son las siguientes:

- Independientemente de la forma asignada al móvil, se asumirá, para fines de la definición de una trayectoria sin colisiones, que se trata de una esfera de radio igual a la separación entre su centroide y la más lejana de sus aristas.
- La superficie de trabajo tendrá dimensiones finitas que el usuario no podrá modificar.
- El móvil no posee la capacidad de almacenar (aprender) rutas ya recorridas.
- El desplazamiento del móvil se realiza entre celdas adyacentes. No realiza movimientos diagonales.
- La aplicación puede ejecutarse bajo los siguientes sistemas operativos: Windows 98, Windows Millenium Edition, Microsoft Windows NT 4.0, Microsoft Windows 2000 y Microsoft Windows XP.

2. Descripción de la Aplicación

Una *búsqueda* es un proceso que trata de encontrar una secuencia lógica de estados que le permita alcanzar una meta. Dicho proceso involucra el ignorar alternativas que se le presentan y encontrar diversas secuencias de acciones posibles que conduzcan a la mejor acción.

Algunas aplicaciones reales que involucran el uso de algoritmos de búsqueda son: determinación de una trayectoria, problemas de viajes turísticos, distribución en la VLSI (muy alta escala de integración), robot móvil, secuencia de un ensamble, etc.

La aplicación a desarrollar brinda solución a uno de los problemas reales anteriormente mencionados, la planificación de trayectorias para objetos móviles. Específicamente, permite conocer la utilidad del algoritmo A* para solventar este tipo de problemas.

La planificación de trayectorias para objetos móviles involucra una serie de de etapas:

1. Planificación de la ruta
2. Construcción de un grafo de conectividad
3. Ejecución de un algoritmo de búsqueda al grafo de conectividad para encontrar la meta.

A dicho proceso le ha sido agregado, con fines didácticos, la graficación de la ruta en dos y tres dimensiones.

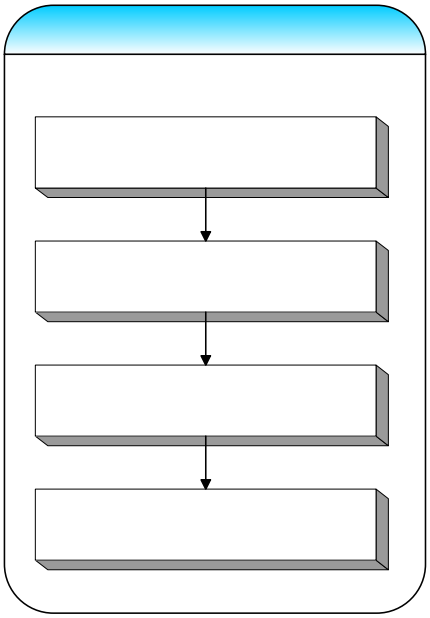


Figura 1. Estructura lógica de Aplicación

2.1 PLANIFICACIÓN DE LA RUTA

Existen diferentes clases de algoritmos que tratan sobre la planificación de trayectorias. Dichos algoritmos manejan el *espacio de configuración C* del objeto móvil, en donde se describen todas las posibles configuraciones del entorno.

El espacio continuo *C* dentro de la aplicación se encuentra representado por un cuadro libre, en el cual, el usuario puede colocar obstáculos y configurar las posiciones de inicio y fin de la trayectoria.

La aplicación desarrollada utiliza el método de *descomposición en celdas* que consiste en dividir el espacio continuo en una cantidad finita de celdas, que da como resultado un problema de búsqueda discreta.

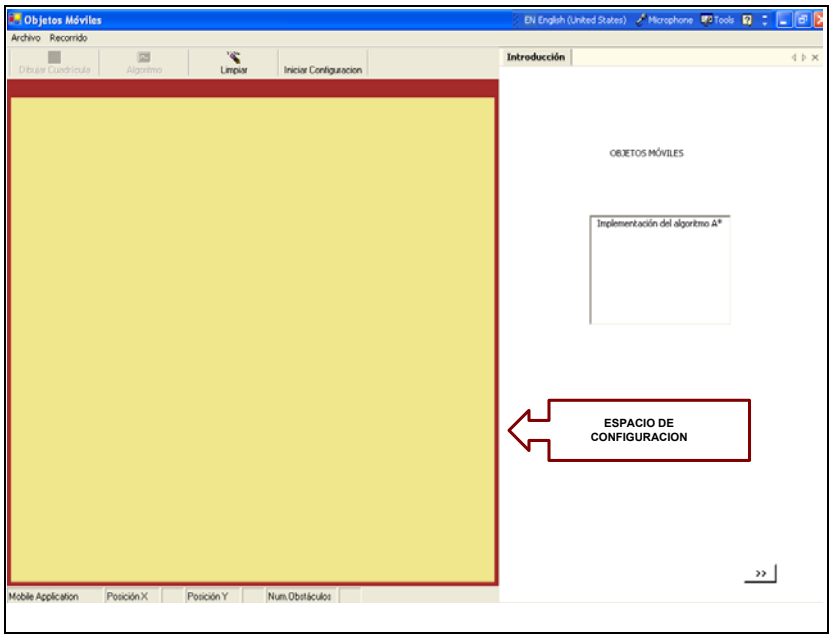


Figura 2. Espacio de configuración en la aplicación

Siendo *C* el espacio de configuración, el método se describe a continuación:

1. Dividir el espacio de configuración *C* en figuras geométricas simples. La división se ha realizado en celdas de dimensión x,y correspondiente a la dimensión del móvil (ancho,largo). Esta configuración es visible mediante una cuadrícula dibujada en el espacio de configuración.
2. Definir en qué celdas se encuentran las posiciones de inicio y fin.
3. Asignar información binaria de estado a cada celda para indicar si está siendo ocupada por un obstáculo o no. El estado puede ser "ocupada" (1) por un obstáculo o "libre" (0).
4. Creación del espacio libre *L* dentro del espacio de configuración, perteneciendo a *L* aquellas celdas libres de obstáculo.

2.2 CONSTRUCCIÓN DE GRAFO DE CONECTIVIDAD

La segunda etapa consiste en la construcción de un grafo no dirigido asociado a la descomposición en celdas descrita en la parte anterior, de tal forma, que los nodos del grafo serán cada una de las celdas, existiendo una arista entre dos celdas si y solo si son adyacentes y libres de obstáculos.

La siguiente figura muestra la descomposición en celdas realizado sobre el espacio de configuración. El grafo de conectividad es construido a partir de las celdas que no se encuentran ocupadas por obstáculos.

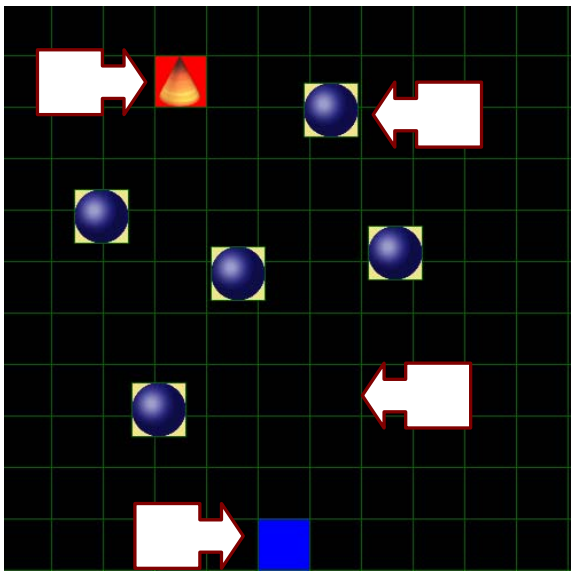


Figura 3. Configuración de espacio y descomposición en celdas

Durante esta etapa, la aplicación brinda la opción de exportar configuraciones del espacio del móvil basándose en una estructura de grafo, con la finalidad que una aplicación externa pueda tomarlas y desarrollar su solución en base a un algoritmo de búsqueda en grafos. La aplicación externa deberá almacenar la ruta solución en un archivo de texto según el formato establecido por la aplicación. Una vez resuelta la trayectoria, la aplicación podrá tomar un archivo de texto con la ruta solución y la graficará en dos y tres dimensiones.

2.3 ALGORITMO DE BÚSQUEDA

Una vez creado el grafo de conectividad, se empleará el algoritmo de búsqueda A* para establecer la ruta óptima para el objeto móvil.

Definición

El algoritmo A* fue creación de Hart, Nilsson y Raphael en 1968 (Russell y Norvig , 1996:123). Este algoritmo es utilizado para la búsqueda en grafos y forma parte de la búsqueda heurística implementada en inteligencia artificial, para la resolución de problemas.

Se define al algoritmo A* como una búsqueda preferente por lo mejor, en la que se utiliza una función *F* como la función de evaluación y una función *h* aceptable. De tal forma que la función F se puede definir de la siguiente forma:

$$f_{(n)} = g_{(n)} + h_{(n)}$$

Variables	Representada por:
$g_{(n)}$	La distancia (costo) que existe entre una celda y otra (de nodo a nodo), que forman parte de la trayectoria del objeto móvil.
$h_{(n)}$	La distancia en línea recta que existe entre cada celda (nodo del grafo) y la ubicación de la meta (nodo final), que forman parte de la trayectoria del objeto móvil.

Tabla 1. Función heurística

El pseudocódigo asociado a la búsqueda A* se detalla a continuación:

Función BÚSQUEDA A*(problema) responde con una solución o falla

Responde con BUSQUEDA-Preferente-por lo mejor(problema,g+h)

- **Propiedades**

A* como todo algoritmo tiene sus desventajas (utiliza más recursos de memoria). Sin embargo, el tiempo en realizar la búsqueda es menor, puesto que combina las ventajas de la búsqueda avara (reduce al mínimo el costo de la meta –h-) y la búsqueda por costo uniforme (reduce al mínimo el costo de la ruta –g-).

Es importante conocer algunas de las propiedades del algoritmo A*, entre las que se incluyen su calidad óptima, integridad y complejidad.

1. CALIDAD ÓPTIMA

A* es un algoritmo óptimamente eficiente, es decir, ningún otro algoritmo asegura que expandirá menos nodos que A*. Dicho algoritmo, al buscar la solución, óptima, expande todos los nodos en los contornos situados entre la raíz y el contorno meta.

2. INTEGRIDAD

El algoritmo A* es completo en grafos finitos, realizando una expansión de los nodos en orden creciente, hasta que encuentre el estado meta o solución.

3. COMPLEJIDAD

La complejidad del algoritmo A* radica en la obtención de una adecuada heurística *h* para el problema que se quiere tratar. La principal desventaja de este algoritmo consiste en el espacio en memoria utilizado, ya que A* guarda en memoria todos los nodos generados.

2.4 REPRESENTACION GRÁFICA

La aplicación contempla la representación gráfica del ambiente y los obstáculos que rodean al móvil y el movimiento de éste a través de la ruta planeada. La representación gráfica se realizará en dos y tres dimensiones.

- **Graficación en dos dimensiones**

Este tipo de graficación incluye el espacio de configuración, costos, obstáculos y el móvil.

Como parte de configuración del entorno del móvil, el usuario puede considerar costos de desplazamiento entre una celda y otra; de tal forma, que es posible elegir entre una serie de materiales y tipos de terreno a ser tomados en cuenta por el algoritmo A*. Dichos materiales son representados mediante diferentes colores y pueden ser aplicados al entorno haciendo click sobre una celda del espacio. Entre los materiales se encuentran: hielo, cemento, alfombra, piedra, etc. Cada uno de ellos tiene un costo asociado iniciando en un tipo de terreno liso y plano con un costo de uno, hasta un terreno con mayor dificultad para ser atravesado, el terreno pedregoso con un costo de diez.

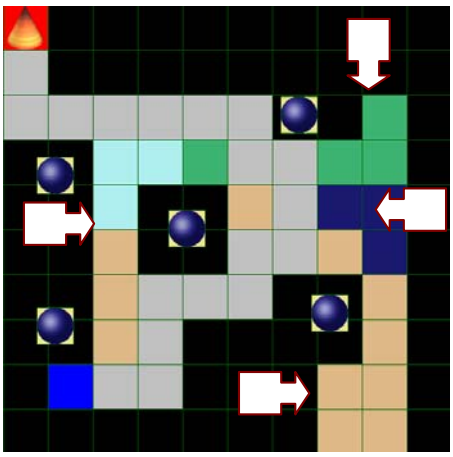


Figura 4. Costos asociados a celdas del entorno

El usuario tiene la posibilidad de elegir entre una serie de obstáculos a ser arrastrados en el espacio de configuración. La dimensión de los obstáculos puede ser elegida de acuerdo al ancho y largo de su base, de tal forma, que la imagen colocada en el espacio de configuración, corresponde a la vista de planta del obstáculo.

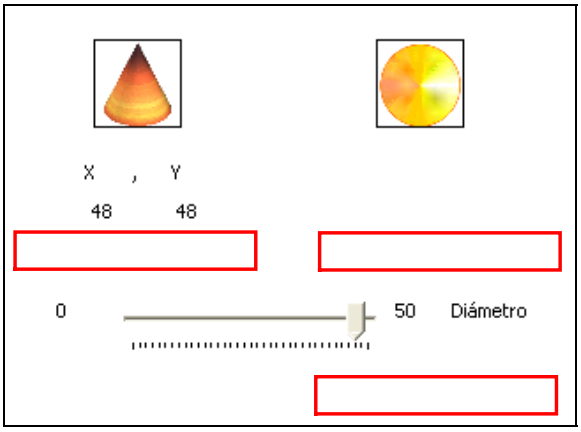


Figura 5. Obstáculos

Una vez colocados los obstáculos requeridos en el espacio de configuración, el usuario puede elegir el objeto móvil, la posición de inicio y fin de la trayectoria.

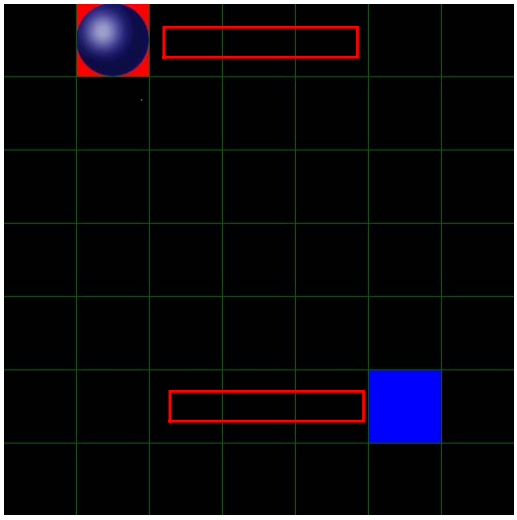


Figura 6. Posiciones del móvil

El objetivo principal de la aplicación, consiste en aplicar el Algoritmo A* al problema de búsqueda de trayectorias, es por ello, que se ha dado énfasis especial en su graficación.

Cuando es generada la ruta, se tiene la opción de graficar únicamente la ruta solución o desplegarla con detalles para una mejor comprensión.

La vista detalles brinda especificaciones sobre las posibles posiciones que va tomando el móvil antes de decidir una ruta en particular. La vista de detalles incluye:

1. Celdas Sucesoras

   Se consideran celdas sucesoras a aquellas en posición superior, inferior, derecha e izquierda de la celda actual. Son celdas adyacentes al nodo actual en el que se realiza la búsqueda y candidatas a ser celda actual. Se elige la de menor costo (f).

2. Celda Actual

   Nodo seleccionado de menor costo (f). Puede o no formar parte de la ruta. Se considera una celda exploratoria para encontrar la solución.

3. Celda solución

   Celdas que forman parte de la solución. Ruta óptima.

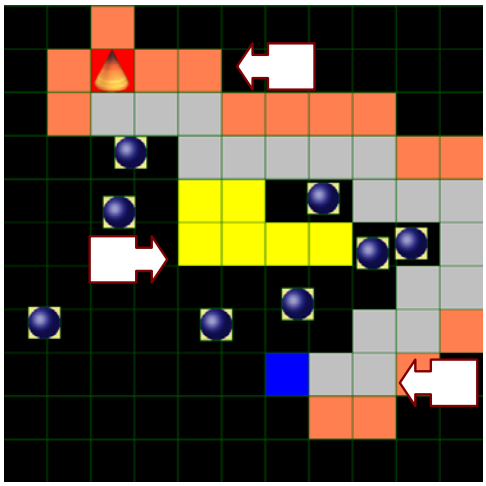


Figura 7. Vista detalles

Graficación en tres dimensiones

La graficación en tres dimensiones pretende que el usuario posea una visión más real del desplazamiento del móvil. Para llevar a cabo esta parte, se utilizan las librerías de OpenGL y las librerías de enlace dinámico CSGL[1] en la creación de un ambiente en donde se ubica al objeto móvil.

Para la simulación en tres dimensiones, el entorno se convierte en un problema estándar en dos dimensiones por medio del mapa de celdas generado a partir del método de planificación descrito en las partes anteriores.

Una vez generada la ruta solución por el algoritmo, el entorno se dibuja en tres dimensiones y se presentan las siguientes opciones para su simulación:

- **VER SOLUCIÓN**

Esta opción realiza la simulación del recorrido del móvil a través de la trayectoria encontrada por el algoritmo A*. Existen las opciones de simulación únicamente de la trayectoria o de recorrido con detalles. Esta última opción grafica las celdas sucesoras, actuales y la solución.

- **VISTA**

Esta opción presenta diferentes configuraciones de visualización durante el recorrido del móvil. Dichas configuraciones hacen referencia a la posición de la cámara utilizada en la aplicación. Los valores tomados en consideración corresponden al ángulo vertical de la cámara, ángulo horizontal y distancia de la cámara con respecto al objeto que se observa.

[1] CsGL (C Sharp Graphics Library) . Conjunto de librerías de OpenGL para uso del .Net Framework. Proveen soporte para OpenGL 1.1 – 1.4 .

Las vistas disponibles se enumeran a continuación:

- Aérea : se visualiza el entorno del móvil desde cierto ángulo de elevación con respecto al plano XZ.

- Planta: la escena es visualizada desde un ángulo de 90° con respecto al plano XZ.

- Móvil: vista del recorrido del móvil en primera persona. La escena es visualizada al nivel del plano XZ.

- Libre: esta opción brinda la capacidad al usuario de manipular los valores de cámara, tales como ángulo horizontal, ángulo vertical, distancia y desplazamiento a través de los ejes X y Z.

- **TERRENO**

Se presentan diferentes opciones de graficación del espacio de configuración del móvil. Las opciones se describen a continuación:

- Cuadrícula y costos: el espacio de configuración es visualizado como una cuadrícula generada por el método de descomposición en celdas. Los costos pueden ser visualizados mediante diferentes colores correspondientes al tipo de terreno o material elegido.

- Textura y costos: diferentes texturas son aplicadas a cada una de las celdas correspondiendo al tipo de costo asociado a ellas.

- Textura: una textura es aplicada a todo el entorno de configuración.

- **POSICIÓN MÓVIL**

Son presentadas las diferentes posiciones del móvil a medida que la trayectoria solución es recorrida. Las posiciones corresponden a su posición en los ejes X y Z.

- **VELOCIDAD**

Esta opción permite regular la velocidad del recorrido del móvil. Las velocidades son alta, media y baja.

- **LUZ**

Controla la iluminación de la escena generada en tres dimensiones. Las propiedades que pueden ser modificadas se detallan a continuación:

- Luz ambiental: controla los colores rojo, verde y azul que son utilizados por la iluminación ambiental. La iluminación ambiental corresponde a un tipo de luz dispersa en al ambiente, cuya fuente es imposible de determinar.

- Luz difusa: permite regular la intensidad de los colores rojo, verde y azul empleados por la iluminación difusa.

- Luz especular: esta opción regula los niveles de rojo, verde y azul pertenecientes a la iluminación de tipo especular. La iluminación especular se caracteriza por provenir de una dirección y tiende a reflejarse en una dirección preferida.

- Posición: controla la posición de una fuente de luz.

La siguiente figura presenta una configuración del entorno del móvil, en el cual se ha elegido la vista aérea y terreno con texturas.

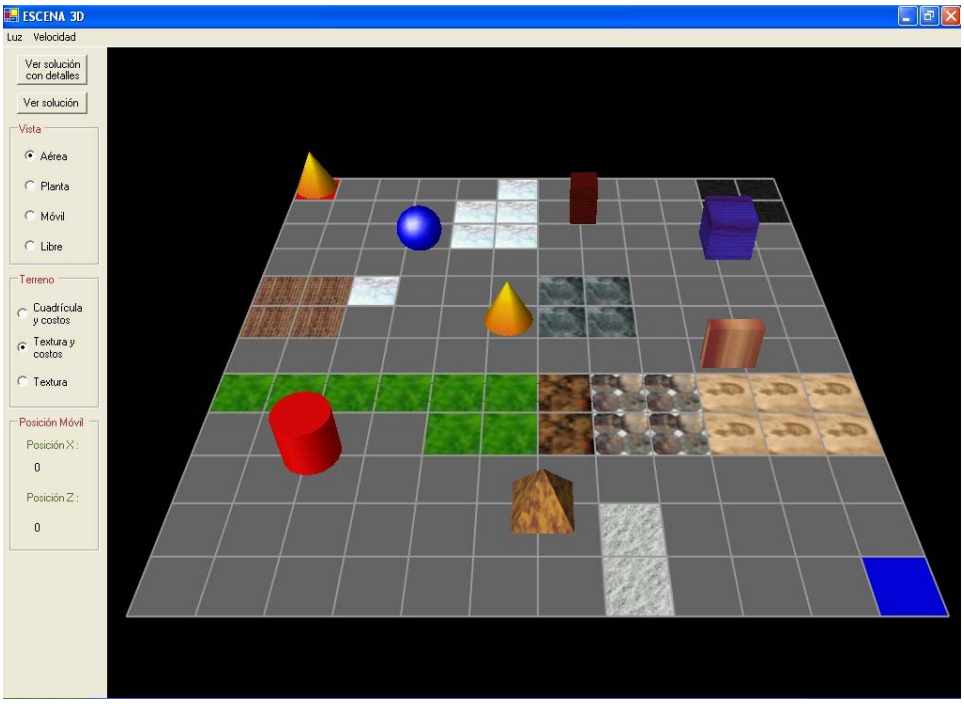


Figura 8. Escena en tres dimensiones

3. Diseño de la Aplicación

3.1 ESTRUCTURA DE APLICACIÓN

- **Estructura**

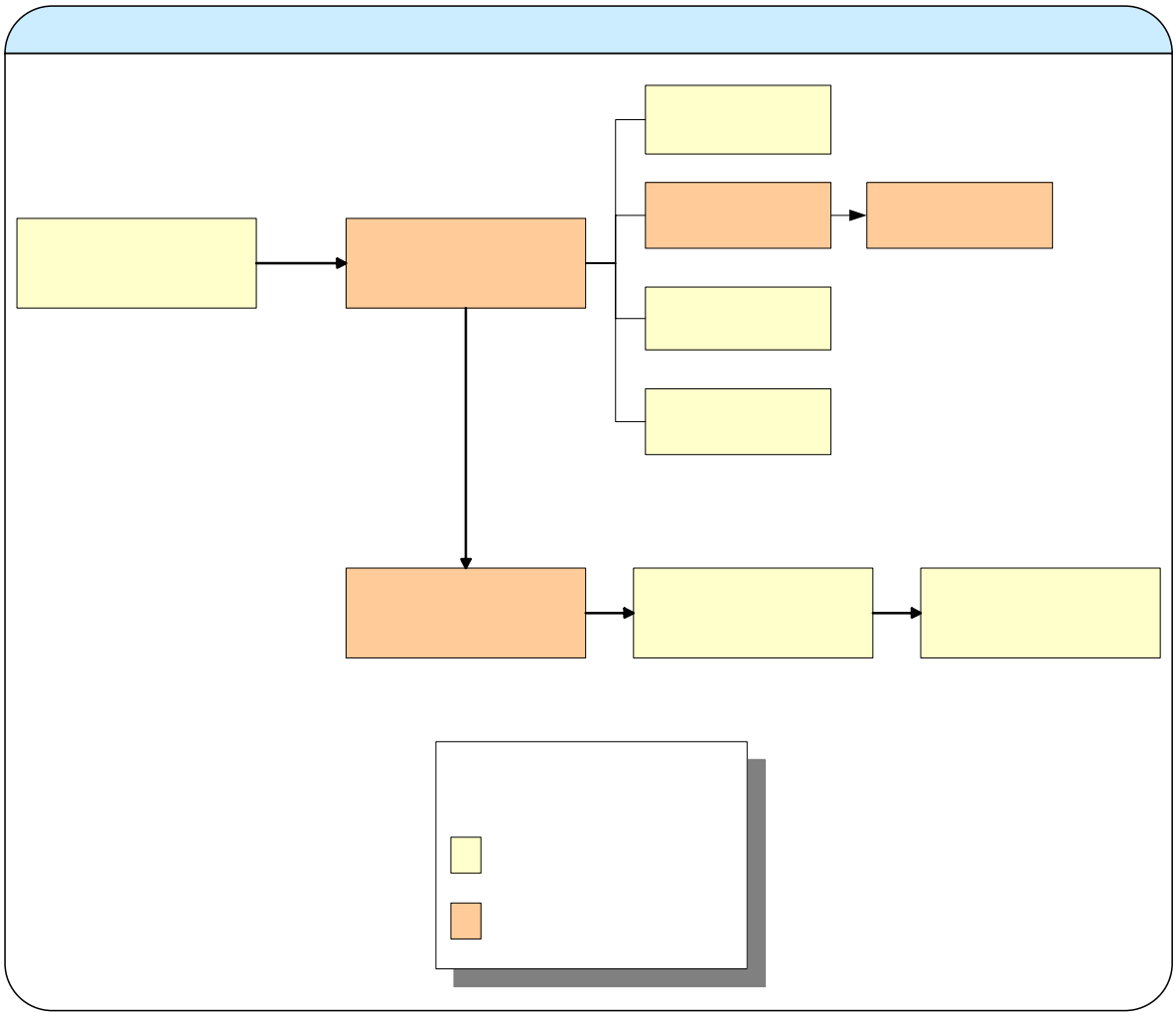


Figura 9. Estructura de la aplicación

El diagrama anterior muestra la estructura interna de la aplicación a desarrollar. En éste se representan la serie de procesos o procedimientos que se seguirán para encontrar una solución al *problema de planificación de trayectorias de un objeto móvil*. Dicha estructura toma como base las características generales que se han observado en los algoritmos de búsqueda y métodos de planificación de robots.

La estructura de la aplicación emplea tres componentes que pueden ser considerados como los mecanismos principales para realizar las distintas etapas de la planificación de la trayectoria del móvil, estos son: *el método de descomposición en celdas, el algoritmo de búsqueda A*, y el operador del problema (clase grafo)*.

Con el objetivo de demostrar las características del algoritmo y como se puede llevar a la practica con la planificación de trayectorias de objetos móviles, se utiliza una representación en dos y tres dimensiones del espacio por donde se desplazara el móvil. Dicha representación servirá para analizar la solución encontrada por el algoritmo.

Cabe señalar que todos los componentes de la aplicación, excepto la graficación en tercera dimensión, se encuentran formados por un conjunto de tipos de datos y funciones soportadas por el lenguaje de programación Visual Basic .NET. Para la representación en 3D del espacio por donde navegará el móvil, se utilizan tipos de datos y funciones definidas en OpenGL (librerías estándar para programación en tercera dimensión).

A continuación se describen cada uno de los componentes de la estructura interna de la aplicación:

Método de planificación de descomposición en celdas

Tal como se mencionó en el apartado de la descripción de la aplicación, este método se encarga de dividir en celdas de igual tamaño el espacio con obstáculos por donde se desplazará el móvil. El procedimiento encargado de realizar este proceso es *Crear_Cuadricula* que se encuentra dentro del módulo "Definiciones.vb".

La estructura y definiciones de los elementos que forman parte de este componente se muestran a continuación:

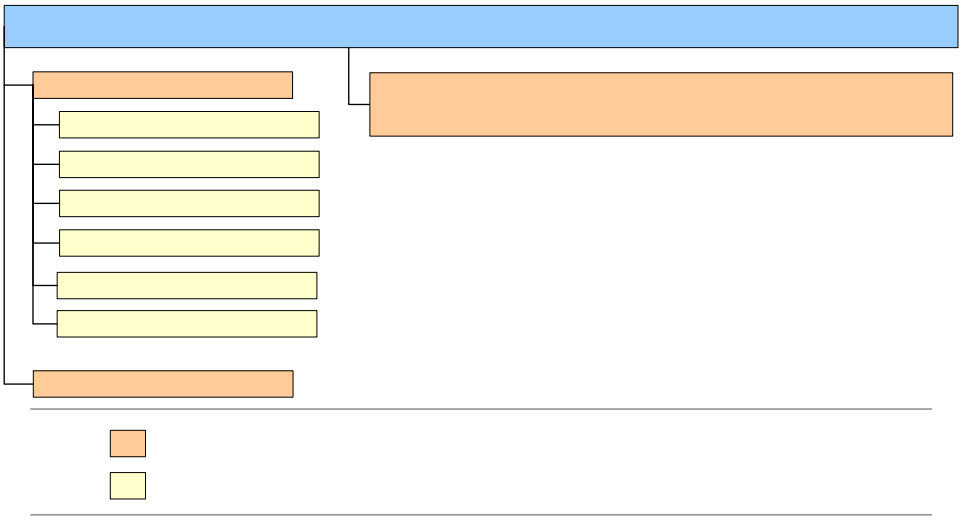


Figura 10. Módulo Definiciones

Los parámetros recibidos por el procedimiento *Crear_Cuadricula* se detallan a continuación:

a) *X1F,Y1F*: es la posición desde la cual se comenzara a dividir el espacio donde se desplazara el móvil (*control panel* de Visual Basic .NET).

b) *Ncelli*: es la cantidad de celdas horizontales que cubren el ancho máximo del espacio por donde se desplazara el móvil. Tal valor se obtiene de dividir el ancho del control panel entre el ancho del objeto móvil (*wMovil*).

c) *Ncellj*: es la cantidad de celdas verticales que cubren la altura máxima del espacio por donde se desplazara el móvil. Tal valor se obtiene de dividir la altura del control panel entre el ancho del objeto móvil.

d) *Costo*: almacena el costo de acuerdo a los valores asignados a cada tipo de material o terreno del entorno.

e) *WCelda*: es el ancho de la celda que es equivalente al ancho del móvil (*wMovil*).

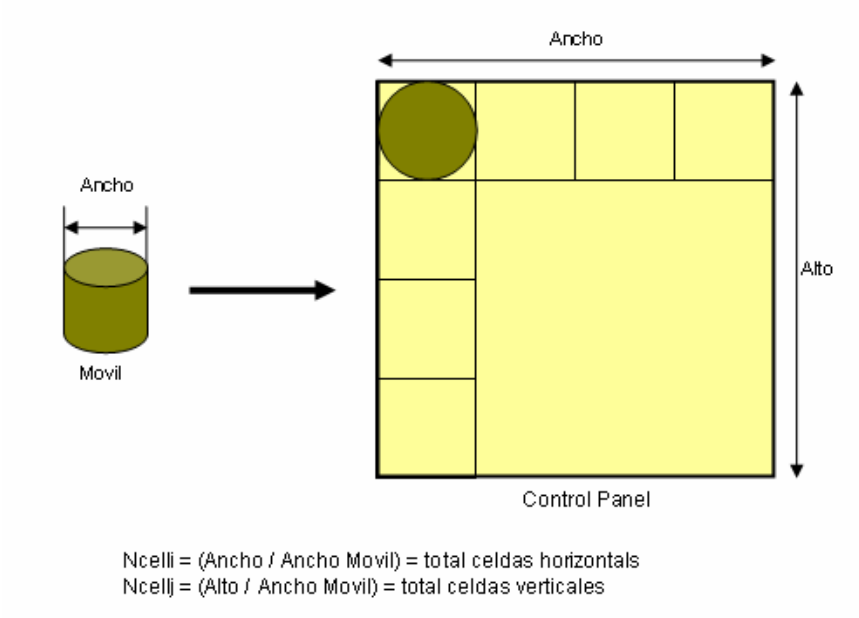


Figura 11. Análisis del procedimiento Crear_Cuadrícula

La información de cada celda derivada de la discretización del espacio donde se encuentra el móvil, se almacena en la estructura llamada *cuadros*. Dicha estructura almacena la posición X,Y a la que pertenece la celda dentro del espacio, el ancho de la celda, costo y el estado de dicha celda (ocupada o libre) tal como se muestra en la figura 9.

- **Problema (Búsqueda sobre un grafo de visibilidad)**

Es importante aclarar, que al aplicar el método de descomposición en celdas para planificar la trayectoria del móvil, *el problema se reduce a una búsqueda sobre un grafo de visibilidad* (conectividad entre celdas). Entonces es posible aplicar un algoritmo de búsqueda sobre grafos, que en este caso, se utiliza el algoritmo A* para encontrar una ruta por donde se desplazará el móvil. Tal ruta se considera libre de obstáculos y óptima.

Las figuras 11 y 12 muestran el proceso de reducir la búsqueda a un grafo de conectividad a partir del método de planificación de descomposición en celdas descrito en el apartado anterior. La primer figura muestra el espacio de configuración dividido en celdas, encontrándose ocupadas por obstáculos las celdas 2, 3, 4, 7 y 8.

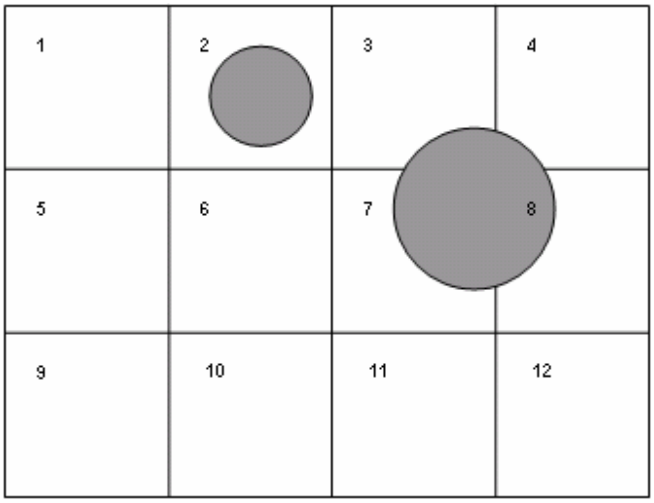


Figura 12. Espacio dividido en celdas donde se encuentra el móvil

La figura 12 muestra el grafo de conectividad generado por la función *Crear_Cuadrícula* descrita en el apartado anterior. Se puede apreciar que dicho grafo está conformado por aquellas celdas libres de obstáculos.

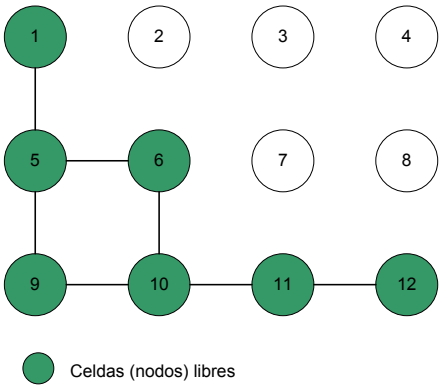


Figura 13. Grafo de conectividad asociado con la descomposición en celdas

Para la construcción de este grafo, la aplicación se basa en funciones y rutinas utilizadas en lenguajes de programación para manipular estructuras de datos como listas[2], grafos y arreglos (vectores). Entre dichas funciones se encuentran: agregar nodo, eliminar nodo, recorrer nodos de una lista, etc.

En general, el recorrido de un grafo se realiza utilizando una estructura auxiliar de tipo lista; y el algoritmo A* no es la excepción. A continuación se muestra la estructura del nodo perteneciente a dicha lista auxiliar llamada *NodoL.*

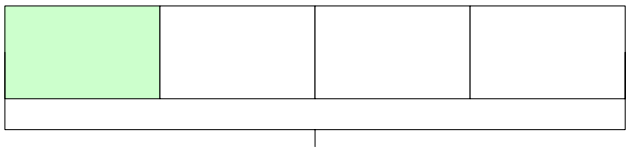


Figura 14. Estructura parcial del nodo de las listas utilizadas en la aplicación

Cada nodo del grafo es controlado por el elemento "*coordenada* (x, y)" de la estructura del nodo de la figura anterior.

[2] En los lenguajes de programación una *Lista* es considerada como una serie de elementos que tienen una misma estructura. Estos elementos están conectados por un campo en común que sirve como enlace entre cada elemento. Cada elemento individual de la lista es denominado nodo, y cada nodo puede tener uno o más datos del mismo o distinto tipo (enteros, carácter, decimales, etc).

El componente que se encarga de convertir las celdas a nodos del grafo a crear es la función constructora (New) perteneciente a la clase *GrafoB*, que se encuentra dentro del módulo "moduloGrafos.vb".

La estructura y definiciones de los elementos que forman parte de este componente se muestran a continuación:

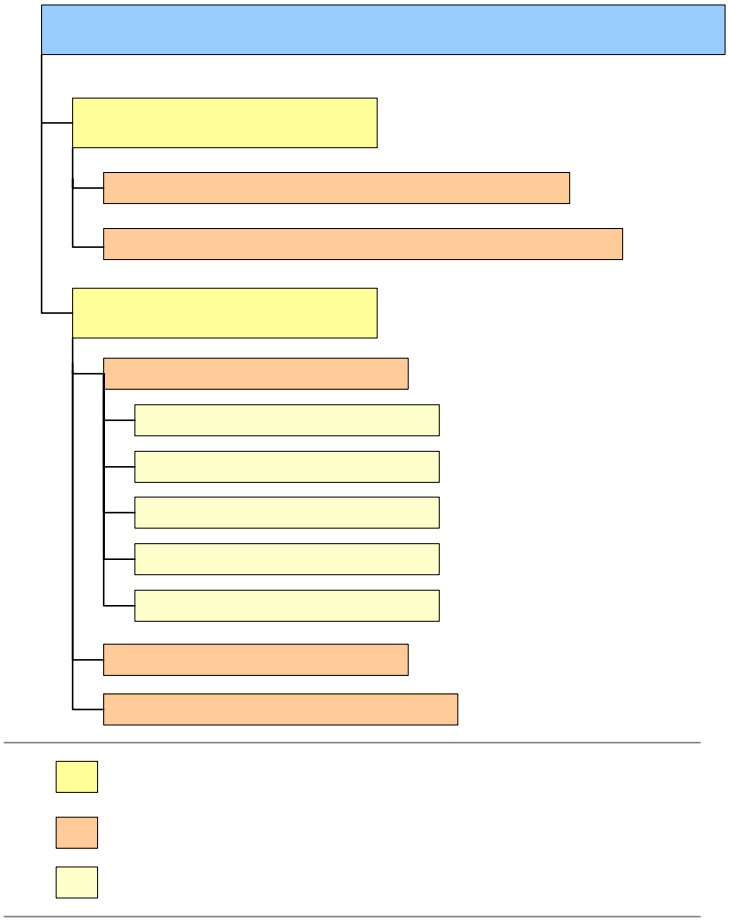


Figura 15. Clase GrafoB

- *Función sucesoresnodo*

  Este método obtiene los sucesores de un nodo que se encuentran adyacentes a él, es decir, se expande el nodo a su derecha, izquierda, el superior e inferior.

Cla
(mo

- *Estructura NodoG*

  Es la estructura base de las listas utilizadas por el algoritmo A* para la manipulación y recorrido del grafo. Posee los siguientes miembros:

  - *nXY :* este elemento controla los nodos pertenecientes al grafo. Contienen la coordenada de la esquina superior izquierda de cada celda del espacio de configuración.
  - *Enlace*: almacena la información de un nodo padre.
  - *G* : almacena el costo que existe entre un nodo y otro.
  - *h:* almacena la distancia estimada entre el nodo y la posición final.
  - *F:* función evaludora que contiene sumatoria de *g* y *h*.

- Variable celdasG

  Estructura que almacena información de las celdas generadas por el método *Crear_cuadrícula.*

- Variable verticeG

  Es la variable utilizada para almacenar los vértices del grafo. Es de tipo *NodoG.*

Es importante destacar que una vez el problema se reduce a una búsqueda sobre un grafo de conectividad, existen elementos que llegan a formar parte del problema, los cuales son:

- **Estado inicial (nodo o coordenada de inicio)**

El móvil necesita saber cuál es el punto de partida desde donde comenzará a planear su trayectoria. Es decir, el algoritmo de búsqueda necesita la coordenada o nodo de inicio donde se encuentra el móvil para saber a partir de donde se comenzará a resolver la búsqueda.

Dicha coordenada de inicio se almacena en la variable *nodoIni* de tipo *coordenada* (Coord: X,Y) que se encuentra dentro del módulo “Definiciones.vb”.

- **Operador (secuencia de acciones posibles: nodos sucesores)**

Durante el proceso de búsqueda, a partir del nodo inicial se generan una serie de nodos sucesores (secuencia de acciones posibles), de ellos se escoge el de menor costo**,** y a partir de él, se genera otra serie de nodos sucesores y así sucesivamente.

Es evidente que parte del problema es encontrar dentro del grafo “nodos sucesores” que permitan pasar de un estado de inicio a otro estado o nodo que este mas cercano del objetivo de la búsqueda: nodo meta. Para ello se implementó el método *sucesoresNodo* que forma parte de la clase *GrafoB*. Este método obtiene los sucesores de un nodo que se encuentran adyacentes a él, es decir, se expande el nodo a su derecha, a su izquierda, el superior e inferior.

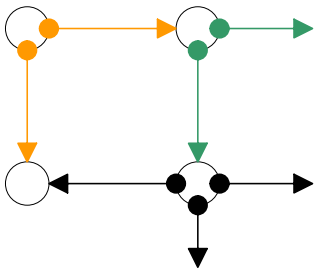


Figura 16. Nodos o vértices sucesores del grafo

En la figura anterior se pueden apreciar los movimientos considerados para obtener los nodos sucesores de un nodo visitado durante el recorrido para ser expandido.

Cabe señalar que si el estado de un nodo es ocupado no se toma en cuenta como sucesor, y también, que para cada nodo a ser expandido tampoco se toma en

cuenta como sucesor el nodo del cual se deriva (nodo padre), tal como se observa en los nodos (1,1) y (1,2) del grafo de la figura 15.

- **Costo de la ruta**

En esta aplicación se considera una superficie plana para el cálculo de la ruta por donde navegará el móvil. Son considerados costos de desplazamiento entre una celda y otra asignándole tipos de terreno a las celdas, es decir, cada celda tiene un costo asociado al material que se ha elegido. Es posible elegir entre diez tipos de terreno, cada uno de ellos con un costo determinado.

Por consiguiente, el costo es considerado como parte del problema y forma parte del cálculo de la ruta durante el proceso de búsqueda. El costo por defecto es igual a 1 y corresponde a un terreno plano y liso.

El costo de la ruta (g) es almacenada en las variables de tipo *NodoG*, definida en la clase *GrafoB.* Dicho valor es calculado por la función *costoNodo* definida en el módulo *MapClass.vb* que se explica más adelante.

- **Evaluación de meta.**

El último componente o información que debe incorporarse al problema es el estado final o nodo (meta) hasta donde debe llegar el móvil. Dicha coordenada de destino se almacena en la variable *nodoMeta* de tipo *coordenada*, ubicada en el módulo *Definiciones.vb*

Esta información es necesaria debida a que en cada desplazamiento de nodo a nodo realizado durante la búsqueda, siempre se evalúa si se ha llegado al final de la ruta. Es el algoritmo A* el encargado de retornar un valor indicando que se ha encontrado el nodo meta durante el proceso de búsqueda.

- **Algoritmo A* (Clase AlgoritmoA)**

Para resolver el problema de la planificación de trayectorias de un objeto móvil, se ha utilizado el algoritmo A* para comprobar y analizar cómo este algoritmo se puede llevar a la práctica a la hora de planificar rutas.

La estructura y definiciones del componente utilizado por el algoritmo A* se muestran a continuación:

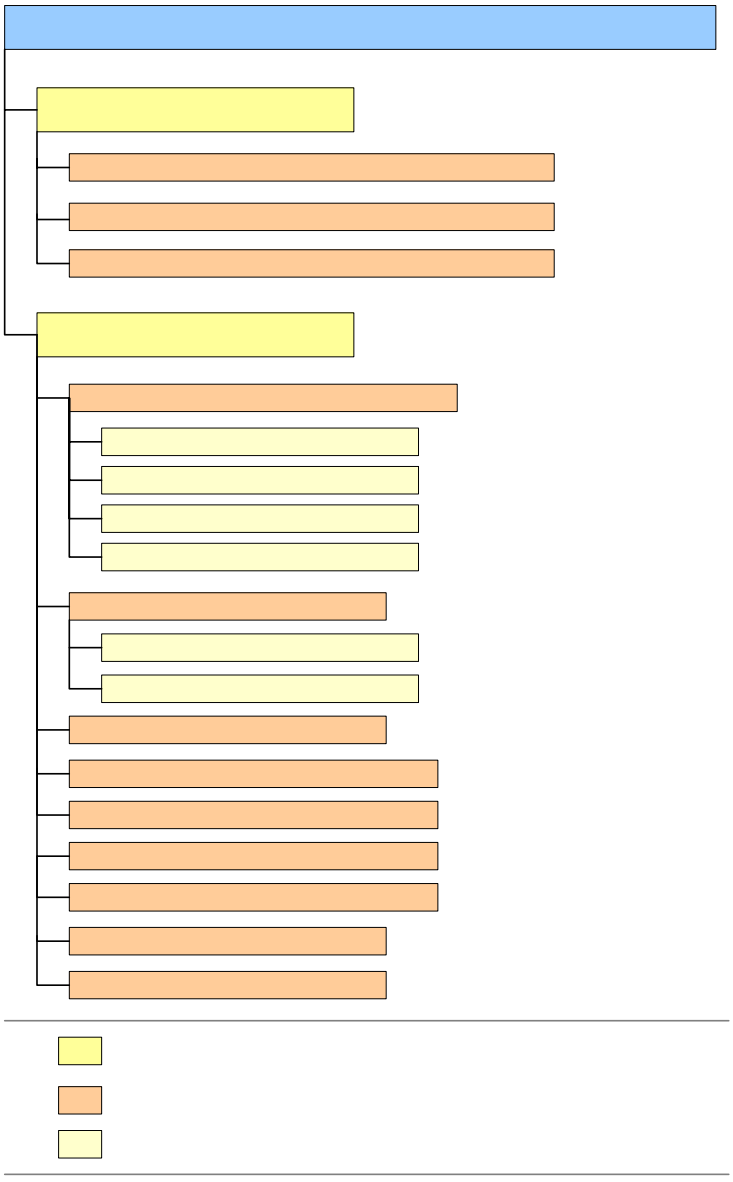


Figura 17. Clase AlgoritmoA

Cla
(m

El grafo creado por los componentes descritos anteriormente es el problema que sirve como entrada para el algoritmo A* y demás algoritmos de búsqueda. El objetivo es encontrar una ruta por donde se desplazará el móvil. Tal ruta se considera libre de obstáculos y óptima.

A continuación se describen las funciones miembro de la clase *AlgoritmoA*:

- Función *configurarIniMeta*
  Esta función realiza la parte inicial del algoritmo. Esta consiste en agregar el nodo inicial a la lista *lstAbierta*. Convierte las coordenadas de la posición de inicio y meta seleccionadas por el usuario, a variables de la clase de tipo *NodoG*. Dichas variables son: *nIni* y *nMeta*.
- Función *buscarSig*
  Esta función es ejecutada dentro de un lazo del algoritmo mientras no se encuentre el nodo meta o no exista solución. Utiliza las listas abierta, cerrada y sucesores para calcular la función evaluadora, verificar enlaces y obtener la ruta solución.
- Función *adicionarNTemp*
  Realiza una copia temporal de un nodo padre durante el proceso de búsqueda.

Los elementos a destacar en la implementación del algoritmo A* se detallan a continuación:

a) Función evaluadora (f)

La idea básica es asumir que se tiene una función de evaluación *f* que ayuda a decidir cual es el mejor nodo para expandir (sucesor) a partir del nodo inicial y siguiente. Los valores mas pequeños para *f* indican los mejores nodos. La función del algoritmo es:

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

g: Es equivalente al costo acumulado de moverse desde el nodo inicial (punto de partida del móvil) hasta otro nodo que es visitado durante el proceso de búsqueda.

Cabe señalar que el costo de moverse de un nodo a otro corresponde al valor elegido para cada celda y no debe confundirse con el valor de "g", ya que éste valor es el costo acumulado de uno en uno desde el nodo de partida hasta otro nodo que sea visitado durante el proceso de búsqueda.

h: En términos generales "h" es la heurística o información estimada para el algoritmo. Dicho de otro modo es la distancia estimada desde un nodo visitado hasta el nodo final o meta.

Este valor se obtiene del cálculo de la *distancia* en línea recta desde la coordenada del nodo visitado hasta la coordenada del nodo meta. Esta distancia se calcula a partir de la siguiente formula matemática:

$$h = \text{distancia} = ((X2-X1)^2 + (Y2-Y1)^2)^{1/2}$$

f: Es la función evaluadora del algoritmo y es igual a la suma de g + h. Es importante destacar que durante el proceso de búsqueda siempre se prefiere el menor valor de "*g*" y por consiguiente el de "*f*" (g+h). Esto nos da la certeza que los nodos a seleccionar para la ruta casi siempre sean los de menor costo y cercanos a la meta.

De lo anterior se deduce que el algoritmo A* combina dos características de algoritmos de búsqueda: menor costo y distancia estimada (búsqueda heurística), y tiene la peculiaridad de que la solución es óptima, es decir casi siempre es la mejor ruta o la más corta. La calidad de la solución obtenida por el algoritmo A* depende en gran medida de la heurística seleccionada. Es decir que si el valor de "*h*" tomado

como referencia es una mala estimación, el algoritmo podría no devolver la ruta más corta (óptima).

Las funciones que calculan los valores de “g” y “h” se encuentran en el modulo *Mapa* como parte de un conjunto de funciones que forman parte del algoritmo A* para el análisis de los nodos. La siguiente figura muestra la estructura del modulo Mapa:

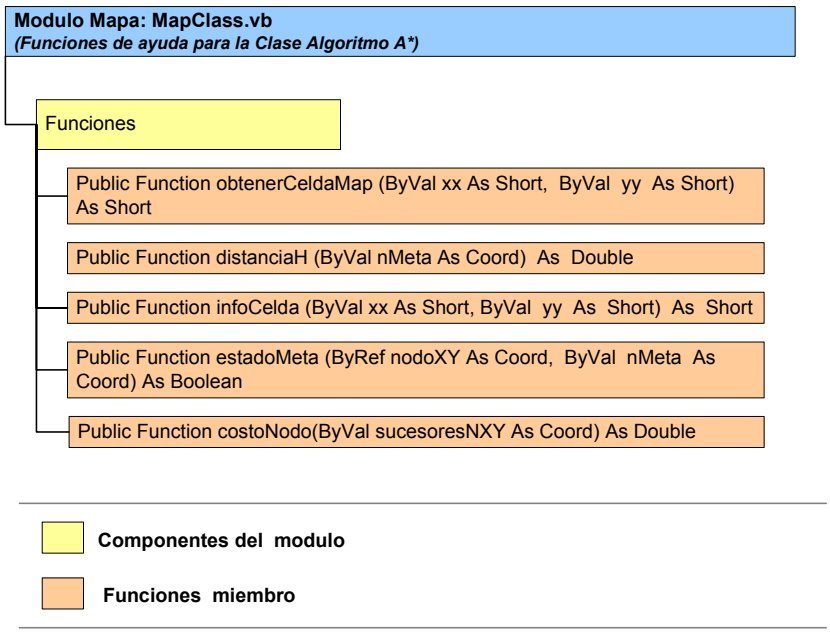


Figura 18. Módulo MapClass

Las funciones asociadas al módulo anterior se detallan a continuación:

- Función obtenerCeldaMap

  Esta función retorna el estado del nodo, es decir, si se encuentra ocupado por obstáculo (1) o libre(0), a partir de su posición horizontal y vertical.
- Función *distanciaH*

  Devuelve la distancia (h) entre un nodo y el nodo destino.
- Función *infoCelda*

  Esta función retorna el número de celda dentro del espacio de configuración a partir de su posición horizontal y vertical

- Función *estadoMeta*

  Esta función verifica si el nodo visitado es igual al nodo meta, devolviendo *Verdadero* si la comparación es satisfactoria ó *Falso* en caso contrario.
- Función *costoNodo*

  Obtiene el costo asociado a una celda dentro del espacio de configuración.

b) Estructura del nodo de las listas utilizadas por el algoritmo

Como se mencionó anteriormente el algoritmo A* utiliza listas para almacenar los valores correspondientes a los nodos del grafo. Basándonos en la función evaluadora utilizada por el algoritmo y la ubicación en el plano de los vértices del grafo, los valores que se almacenan en los nodos de estas listas son:

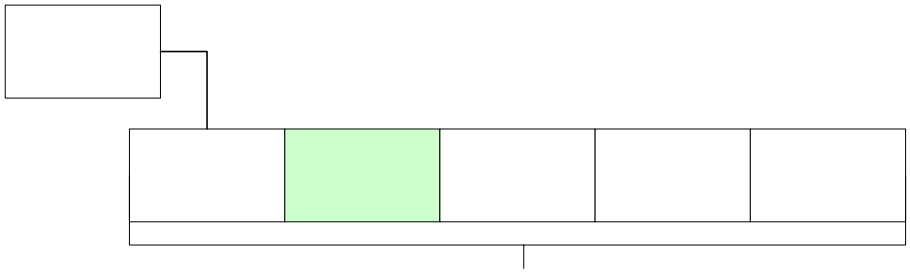


Figura 19. Estructura del nodo de las listas del Algoritmo A*

La aplicación utiliza un módulo llamado "moduloListas.vb". En él se encuentran todas las funciones encargadas de manipular y operar los nodos de las listas creadas para la implementación del algoritmo A*.

La estructura y definiciones de los elementos que forman parte del módulo Listas se muestran a continuación:

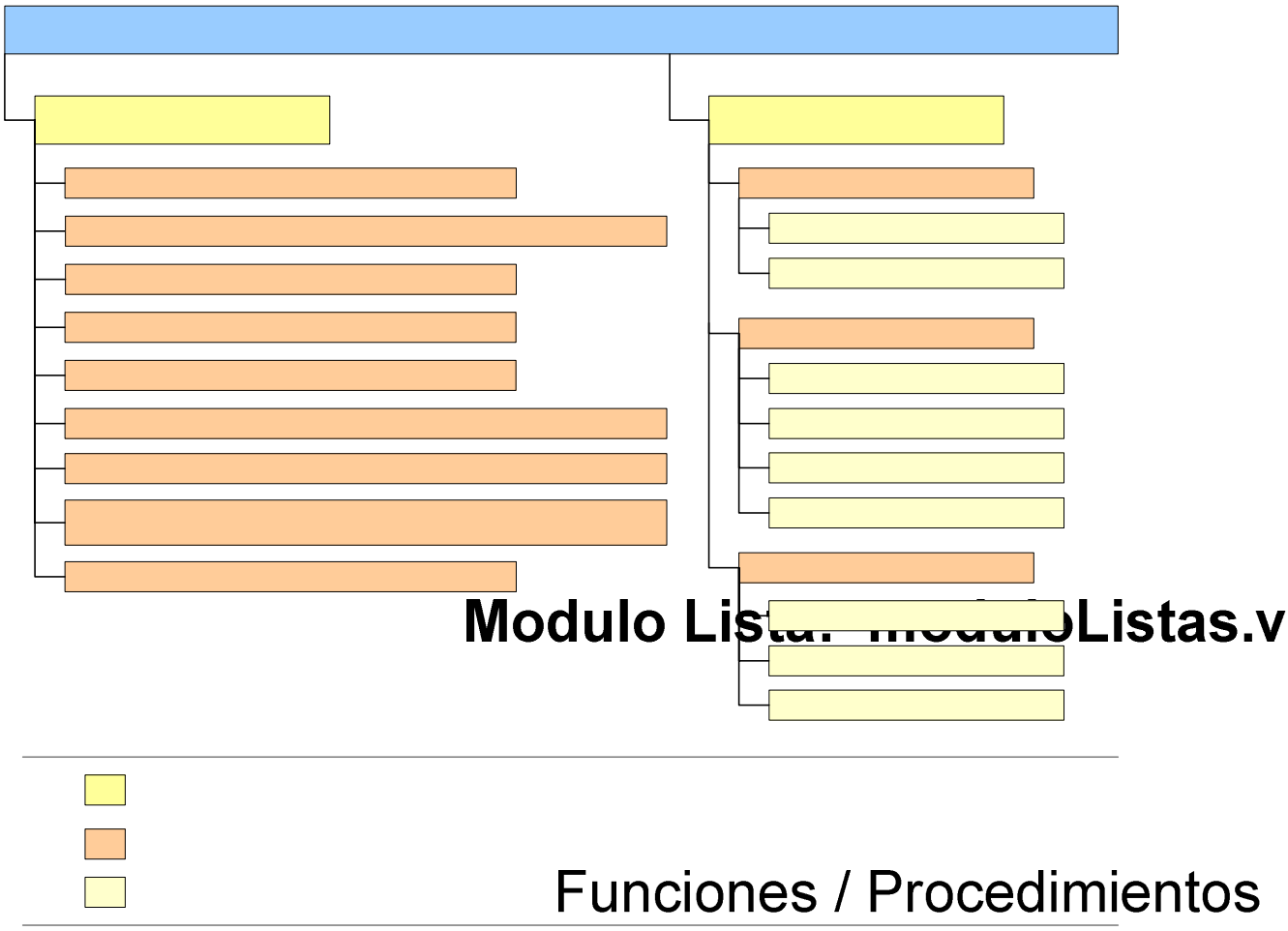


Figura 20. Módulo Lista

c) Listas del algoritmo

Las listas utilizadas por el algoritmo A* se encuentran definidas en la clase *AlgoritmoA* (ver figura 16), las cuales son:

- Lista abierta (lstAbierta)
- Lista de sucesores (lstSucesores)
- Lista cerrada (lstCerrada)

La *lista abierta* almacena todos los nodos que se han generado y a los que se les ha aplicado la función de evaluación, pero todavía no se han expandido sus

sucesores. Los nodos de esta lista son ordenados según el menor valor de *f*. (de menor a mayor costo). Cabe señalar que cuando se escoge un nodo de la lista *abierta* para generar sus sucesores, éste se borra de la lista y pasa a formar parte de la lista *cerrada*.

La *lista de sucesores* almacena temporalmente todos los nodos sucesores que se derivan a partir de un nodo seleccionado para su expansión (el de menor valor de *f: g+h*).

El nodo seleccionado para expandir es el que se encuentra al frente de la lista *abierta* (primer nodo), ya que esta lista tiene ordenados sus nodos de menor a mayor valor de *f*. Cada vez que se expanden los nodos sucesores de un nodo estos son trasladados a la *lista abierta* y ordenados aplicando el mismo criterio.

Es importante destacar que para cada nodo sucesor que es trasladado a la *lista abierta*, siempre se almacena la información de su nodo padre o antecesor (ver figura 18). Esta información es necesaria para poder obtener la solución de la ruta recorriendo los antepasados de cada nodo a partir del nodo meta, si éste es encontrado.

La *lista cerrada* almacena todos los nodos que ya fueron seleccionados para su expansión. (Según el menor valor de *f* encontrado en la lista abierta).

- **Solución (ruta óptima)**

Si el nodo meta es encontrado durante el proceso de búsqueda, quiere decir que se ha obtenido una solución o ruta óptima para el problema de “planificación de la trayectoria” del móvil.

Esta ruta se obtiene recorriendo la *lista cerrada* a partir del nodo meta que es el ultimo en ser seleccionado por el algoritmo. Cabe señalar que es la *lista cerrada* la

que contiene todos los “mejores nodos” (mas óptimos) que fueron seleccionados durante el proceso de búsqueda. Puesto que cada nodo mantiene información de su nodo antecesor, basta con recorrer esta lista desde el ultimo elemento (nodo meta) hasta el primero para calcular dicha ruta.

El proceso detallado del algoritmo A* para obtener la ruta puede ser consultado en el Anexo No.1

- **Representación en 2D**

La aplicación utiliza la representación gráfica en dos dimensiones para una mayor comprensión del funcionamiento del algoritmo A*.

La representación contempla la configuración del entorno del móvil, obstáculos y el resultado generado por el algoritmo.

El usuario elige entre una lista de obstáculos, y arrastra uno de ellos al espacio de configuración. Una vez colocados los obstáculos, se debe elegir el móvil y su posición de inicio y destino.

La representación gráfica del método de descomposición en celdas puede visualizarse en el espacio de configuración mediante su división en una serie de cuadros, graficados mediante el método *dibujar* que pertenece al formulario principal. Dicho método llama al evento *paint* del control panel de *Visual Basic* para actualizar los gráficos. Es de señalar, que el control panel representa el espacio de configuración en donde se desplaza el objeto móvil.

En el momento en que es ejecutado el algoritmo A*, es posible graficar únicamente la ruta solución o desplegarla con detalles para una mejor comprensión. La vista detalles grafica de diversos colores la información: celdas sucesoras, celda actual y celda solución.

Representación en 3D

La graficación en tres dimensiones utiliza las librerías de OpenGL y librerías CsGL que permiten la interacción entre OpenGL y su programación en cualquier lenguaje de programación .NET. La clase *GlView* y el módulo *model* corresponden a objetos pertenecientes a la librería CsGL.

El siguiente diagrama muestra la estructura interna de la aplicación al momento de representar la escena en tres dimensiones.

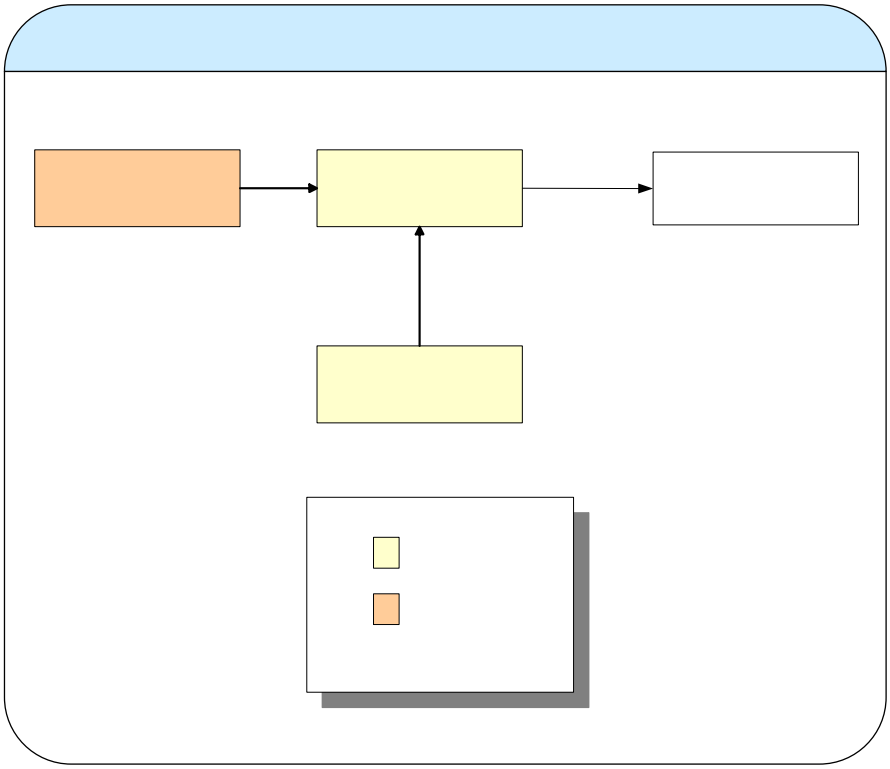


Figura 21. Estructura de graficación en tres dimensiones

El diagrama indica que un modelo es asociado a una vista dentro de la ventana. Dicho modelo contiene los objetos a ser dibujados dentro de la escena. Las funciones que realizan la graficación de los objetos se encuentran en el módulo *objetos*. El conjunto de estas estructuras brindan como resultado la escena y simulación del móvil en tres dimensiones.

- **CLASE GLVIEW**

Esta clase hereda las propiedades de un control de CsGL llamado *OpenGLControl.* Este control es la base de graficación y es asociado al formulario en donde se dibuja la escena en tres dimensiones. En el se eligen los diferentes modelos ha ser graficados.

En la siguiente figura se presentan las funciones asociadas a esta clase:

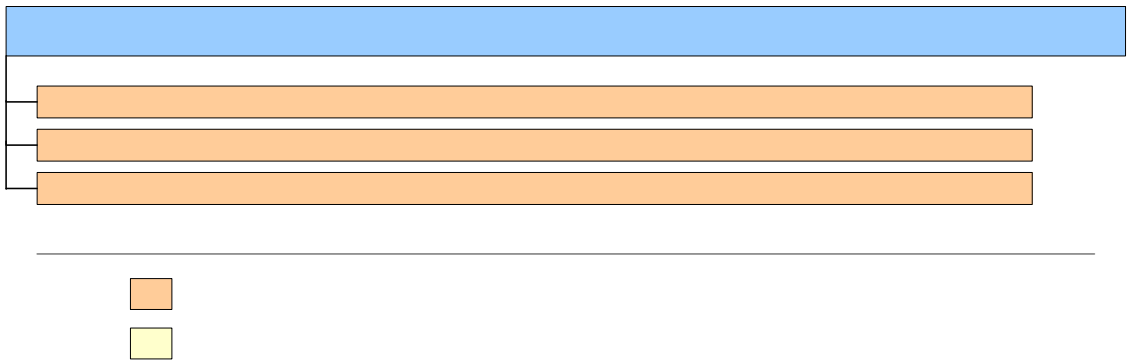


Figura 22. Clase GLView

En la función *glDraw* se define el modelo que será desplegado por este control. La función que dibuja la escena se especifica en el módulo *Model.*

La función *OnSizeChanged* controla el tamaño del control colocado en el formulario. Cada vez que las dimensiones de la ventana cambien, la función reinicia la configuración del sistema de coordenadas.

La función principal en OpenGL de éste método es *glviewport* definida de la siguiente forma:

Void glviewport(GLint, x, GLint y, GLsizei ancho, GLSizei, alto)

Los parámetros *x* y *y* especifican la esquina inferior izquierda de la vista, y el *ancho* y *alto* indican las dimensiones en píxeles. La vista define el área en la ventana en la que OpenGL dibuja.

- **MÓDULO MODEL**

El modelo es un componente de CsGL que consiste en una plantilla en la que se realiza la graficación en tres dimensiones. Provee diferentes métodos para configurar la aplicación, dibujo e interacción con el usuario.

Las funciones pertenecientes a este módulo se presentan en la siguiente figura :

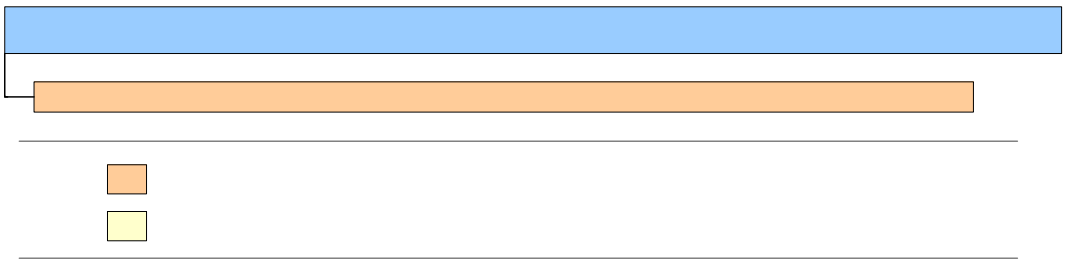


Figura 23. Módulo Model

La función *DibujarEscena* se encarga de detallar el orden en que se dibujan los objetos en la vista. Dentro de esta función se realizan una serie de llamadas a funciones de graficación en el módulo *objetos.*

La siguiente figura muestra el diagrama de bloques en el que se dibujan los diferentes componentes de la escena dentro de la función:

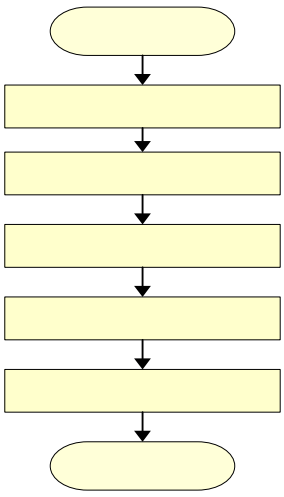


Figura 24. Diagrama de orden de graficación objetos de escena

Módulo: Model.vb

(Modelo de la aplicación)

Public Sub DibujarEsc

- **GRAFICACIÓN DE OBJETOS**

La graficación de objetos dentro de la escena es controlada por la función “DibujarEscena” dentro del módulo “Model”. Esta subrutina llama a cada una de las funciones dentro del módulo “Objetos” dependiendo de las configuraciones del entorno realizadas previamente en dos dimensiones.

La siguiente figura muestra en detalle las funciones de graficación pertenecientes al módulo “Objetos”.

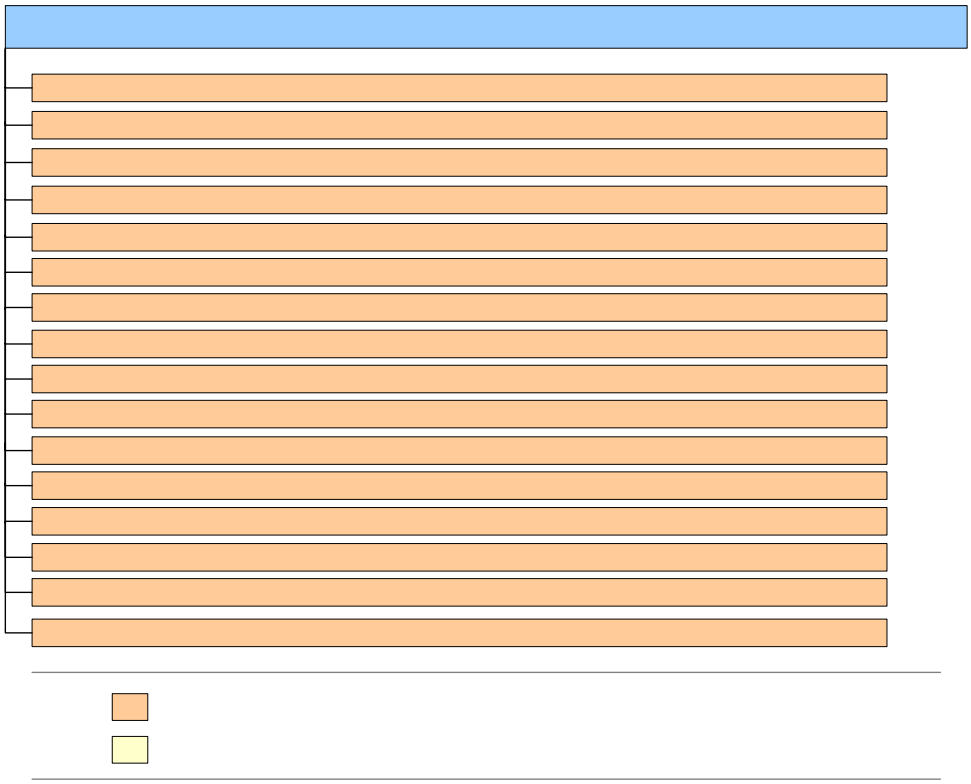


Figura 25. Módulo Objetos

La primera función de graficación que es llamada corresponde a la configuración de la escena (*SetupRC)*. Esta función contiene todos los parámetros de inicialización que deben tomarse en cuenta antes de graficar alguna figura.

Realiza configuraciones de iluminación ambiental, especular, difusa, posición de la luz y combinación de colores.

Para indicarle a OpenGL que realice cálculos de iluminación, se debe llamar a la función *glEnable* con el parámetro *GL_LIGHTING* de la siguiente forma:

Glenable(GL_LIGHTING)

Después de habilitar la iluminación, se realizan las configuraciones de iluminación en el modelo. La iluminación ambiental es configurada mediante la función *glLightmodel* y un arreglo conteniendo los valores RGBA para la intensidad de la iluminación.

La función se define de la siguiente manera:

*Gllightmodelfv(GLenum pname, const GLFloat *params)*

En donde el primer parámetro corresponde al tipo de iluminación ambiental y el segundo parámetro al arreglo con valores RGBA.

La configuración también habilita una fuente de luz llamada GL_LIGHT0 . A dicha fuente de luz es asociado el tipo de iluminación y la intensidad en los colores de la siguiente forma:

- Luz difusa: GL_DIFFUSE
- Luz especular: GL_SPECULAR
- Posición de la fuente de luz: GL_POSITION

La siguiente configuración que se realiza es la comparación en profundidad, permitiendo que los objetos no se sobrepongan entre si. Para ello se debe indicar que no se dibujen aquellos objetos que se encuentren en el mismo nivel o en un menor nivel de profundidad en el eje z que los objetos más cercanos, habilitando dicha comparación con la función : *gldepthfunc(gl_LEQUAL).*

Después de realizar la inicialización de la escena, se procede a colocar los objetos que corresponden a la escena dibujada previamente en dos dimensiones.

La cuadrícula obtenida del método de descomposición en celdas, es dibujada antes de colocar cualquier obstáculo u objeto móvil. El terreno es dibujado dependiendo de la elección que haya realizado el usuario en la barra de configuración de la siguiente forma:

- Terreno cuadrícula y costos: se llama a la función *dibujarcostos*. Esta subrutina es encargada de dibujar una serie de cuadros del tamaño del objeto móvil, aplicando el color de costo asociado a cada celda.

- Terreno textura y costos: a esta opción corresponde la función *dibujarcostostextura.* Esta función es similar a la anterior, con la excepción que asocia una textura a cada tipo de costo y no un color específico.

- Terreno textura: dibuja la cuadrícula del entorno y aplica una textura única a todo el terreno. La función que realiza este trabajo es *dibujarterrenotextura.*

Una vez graficado el terreno, se procede a dibujar al móvil en la posición inicio indicado previamente. Para ello se utilizan dos funciones:

- Graficación de la posición de inicio y destino (*dibujarposiciones)* . Esta función permite identificar las posiciones de inicio y destino mediante colores. La celda inicio es dibujada en color rojo y la celda destino en color azul.

- Dibujar móvil *(dibujarmovil)* es encargada de elegir al tipo de móvil y graficarlo dentro de la escena. El tipo de móvil puede ser una esfera, cubo, cono o cilindro. Dependiendo del tipo de móvil, así es llamada la función de graficación en OpenGL. Esta función también es encargada de la simulación

misma del desplazamiento del móvil, cambiando las posiciones del mismo de acuerdo a las celdas solución obtenidas por el algoritmo A*.

La última función de graficación corresponde a la de *colocarobstaculos.* En ella se obtiene el número y tipo de obstáculos colocados en el espacio de configuración. A partir del tipo de obstáculo es llamada cada una de las funciones de graficación, de la siguiente manera:

Tipo de obstáculo	Función	Textura Asociada	Función OpenGL
Esfera	dibujaresfera		glutsolidsphere
Cubo	dibujarcubo	X	Glbegin(gl_quads)
Cono	dibujarcono	X	Glucylinder
Prisma	dibujarprisma	X	Glbegin(gl_quads)
Cilindro	dibujarcilindro		glucylinder
Pared	dibujarpared	X	Glbegin(gl_quads)
Pirámide	dibujarpiramide	X	Glbegin(gl_triangles)

Tabla 2. Funciones de graficación obstáculos

Cada una de las funciones anteriores es dibujada en la escena en posiciones correspondientes a la configuración en dos dimensiones. Además, reciben parámetros de la dimensión del móvil.

- **MÓDULO VAROPENGL**

Este módulo provee funciones y estructuras de soporte a la graficación de la escena. La siguiente figura muestra los componentes principales de este módulo:

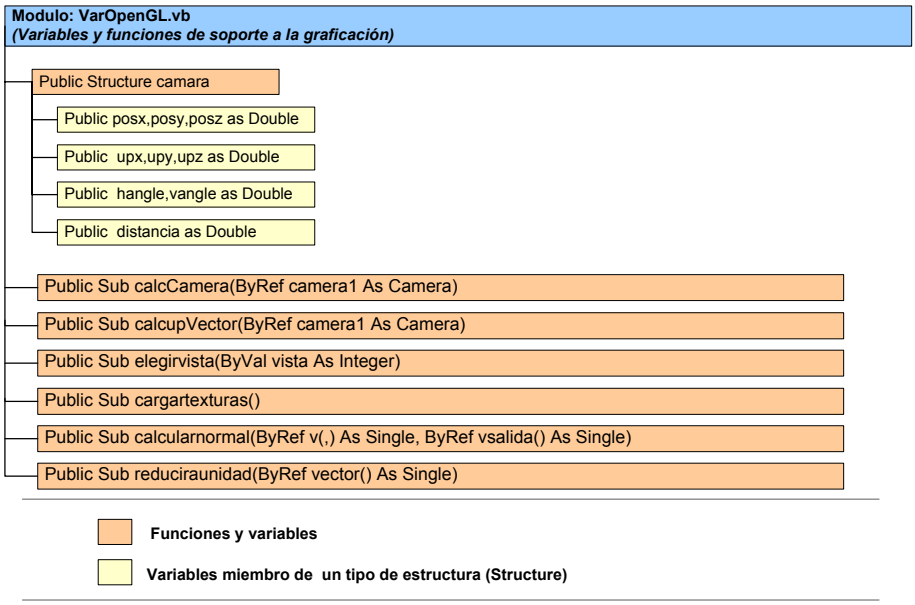


Figura 26. Módulo VarOpenGL

a) Cargar Texturas

OpenGL provee de diferentes funciones para mapear imágenes que contienen texturas a polígonos generados en una escena. La escena generada en la simulación contiene diferentes texturas, es por ello que se ha definido la función *cargartexturas* para cargar dichas texturas durante la ejecución de la aplicación. Una vez cargadas a memoria dichas texturas, ellas son asociadas a los diferentes objetos en el momento de graficarlos.

La función *glGenTextures* indica el número de texturas que serán cargadas en memoria, haciendo referencia a ellas mediante un índice.

Una vez que se cuenta con el número de texturas, se utiliza la función *glBindTexture* para seleccionar el objeto textura.

b) Elegir Vista

La función *elegirvista* configura valores para manipular la cámara. Dichos valores corresponden a las posición de traslación dentro de la escena, distancia de la cámara, ángulo horizontal y vertical del punto de referencia.

c) Normales

El vector normal consiste en una línea desde un punto en un plano imaginario hasta un vértice ubicado en una posición superior en ángulo de 90°. Este vector es utilizado para indicar en que dirección un objeto refleja la luz.

La función encargada de calcular las normales para los polígonos dibujados en la escena es *calcularnormal*. Dicha función obtiene como parámetros un arreglo conteniendo tres vértices de un polígono y un arreglo que contendrá el vector normal calculado. La función de normalización, es decir, la encargada de dividir cada componente de la normal por su longitud, es *reduciraunidad*. De esta forma, el vector normal poseerá sus componentes con una unidad de longitud.

d) Cámara

Para obtener diferentes tipos de visualización del escenario, ha sido necesario crear una cámara que permita obtener diferentes perspectivas. Las funciones asociadas al manejo de la cámara son: *calcCamara* y *calcUpVector*.

Se debe considerar para el diseño de la cámara una región que brinde movimiento alrededor del punto que se está observando. En la siguiente figura se encuentra representada por la esfera de color. El punto *P* es la posición de la cámara dentro de la región de visualización de la cámara.

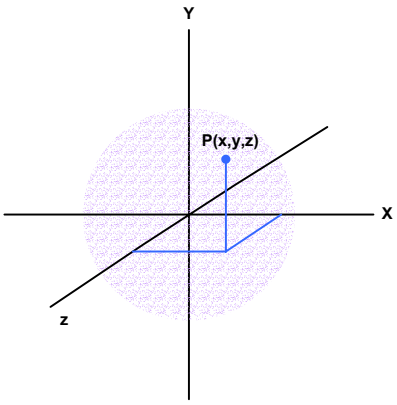


Figura 27. Región de visualización de cámara

El movimiento de la cámara viene dada por la ubicación del punto de observación alrededor de la esfera. Es por ello que deben calcularse tres valores:

- Angulo vertical de rotación de la cámara
- Angulo horizontal de rotación de la cámara
- Distancia de la cámara con respecto al punto de observación.

Estos cálculos deben realizarse en base a la siguiente figura:

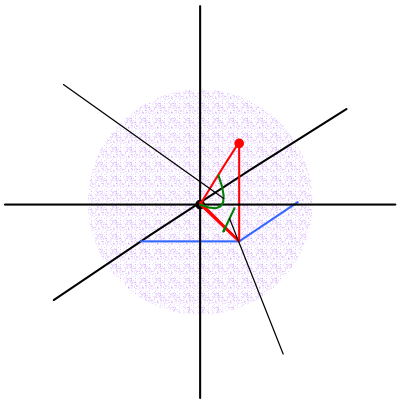


Figura 28. Análisis cámara

1. Cálculo del ángulo vertical

Si se considera al triángulo formado entre el punto *P* y el punto a observar de la siguiente figura:

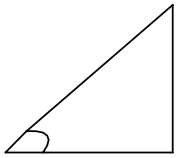


Figura 29. Triángulo de cámara

Por Pitágoras se obtiene la siguiente fórmula para calcular el ángulo vertical:

Vangulo = seno^{-1}(y/radio).

2. Calculando la distancia de la cámara

La distancia ha ser utilizada por la cámara corresponde a la distancia xz (*d)* de la figura anterior. Es posible entonces, utilizar la función coseno para calcular la distancia de la cámara: d = radio x coseno(Vangulo).

3. Calculando los valores X y Z de la cámara

La siguiente figura muestra el plano XZ de la escena dibujada.

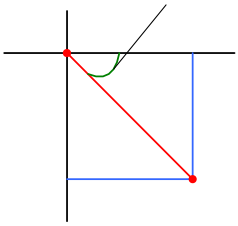


Figura 30. Plano XZ

En base a la figura anterior, es posible deducir las siguientes fórmulas para calcular los valores en *x* y *z.*

X = d x coseno (hAngulo)

Z =d x seno (hangulo)

La cámara utiliza la función de OpenGL *glLookat* de transformación de vista que se define a continuación:

Gllookat(ojox, ojoy, ojoz, centrox, centroy, centroz, verticalx, verticaly, verticalz)

Dicha función define la transformación de la vista basada en la posición del observador, la posición del centro de la escena y el vector vertical desde la perspectiva del observador. Los parámetros se definen a continuación:

- Ojox, ojoy, ojoz: coordenadas del observador
- Centrox,centroy, centroz: coordenadas del centro de la escena que se visualiza.
- Verticalx, verticaly, verticalz: coordenadas que especifican el vector vertical.

Es para uso de la función anterior que resulta necesario crear una estructura (*camara)* que maneje los principales valores de la cámara :

- Distancia: distancia desde la ubicación de la cámara al punto que se está observando.
- vAngulo: ángulo vertical de rotación
- hangulo: ángulo horizontal de rotación
- posx: posición en el eje X de la cámara
- posy: posición en el eje Y de la cámara
- posz : posición en el eje Z de la cámara
- upx: componente X del vector vertical.
- upy: componente Y del vector vertical
- upz: componente Z del vector vertical

El cálculo de estos valores viene dado por las funciones *calcCamara* y *calcUpVector*. Por lo tanto, la función *gllookat* queda de la siguiente forma:

Gllookat(posx,.posy,posz, 0,0,0,upx,upy,.upz)

3.2 DISEÑO DE LA INTERFAZ

FORMULARIO PRINCIPAL

El formulario principal es el punto de partida de configuración del espacio en el que se desplazará el móvil. Sus partes principales se detallan a continuación.

1. Barra de Herramientas

   Contiene opciones generales de la aplicación. En ella se encuentran botones para generar la cuadrícula, generar la ruta, limpiar el espacio de configuración e iniciar el proceso de configuración.

2. Barra de Estado

   En este control se despliega información concerniente a la posición del cursor, cuando éste se posiciona sobre el espacio de configuración; indica el número de obstáculos que han sido colocados sobre el espacio y brinda información general del procedimiento.

3. Espacio de Configuración

   Espacio en el que se colocarán los obstáculos y el objeto móvil. En dicho control se arrastran los obstáculos a la posición deseada desde el formulario de obstáculos. Y una vez elegido el móvil, se elige la posición de inicio y destino.

   En ella se visualiza gráficamente la ruta del móvil con opción de desplegar detalles generados por el algoritmo de búsqueda A*.

4. Guía de procedimiento

   Se ha incluido una guía de uso de la aplicación en la que se van detallando los pasos necesarios para generar la trayectoria de un móvil. Esta guía despliega los formularios de obstáculos, móvil, elección de posición de inicio y fin y el formulario de resultados.

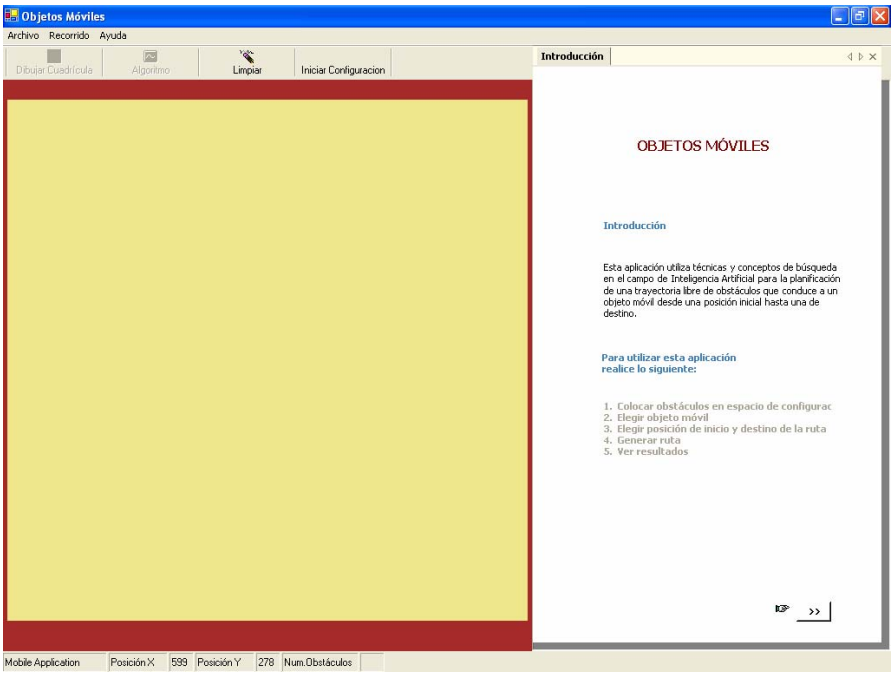


Figura 31. Formulario Principal

FORMULARIO OBSTÁCULOS

Formulario que es desplegado dentro del formulario principal. El usuario puede elegir los obstáculos de una lista limitada. Una vez seleccionado el objeto, se deben elegir las dimensiones de la base del obstáculo. Dependiendo del obstáculo, se eligen dimensiones de ancho y largo ó diámetro.

Dos vistas del objeto son desplegadas: una en la que se puede apreciar el objeto completo y la segunda consiste en la vista de planta del obstáculo.

Posteriormente, el usuario debe arrastrar el obstáculo elegido al espacio de configuración y colocarlo en la posición deseada.

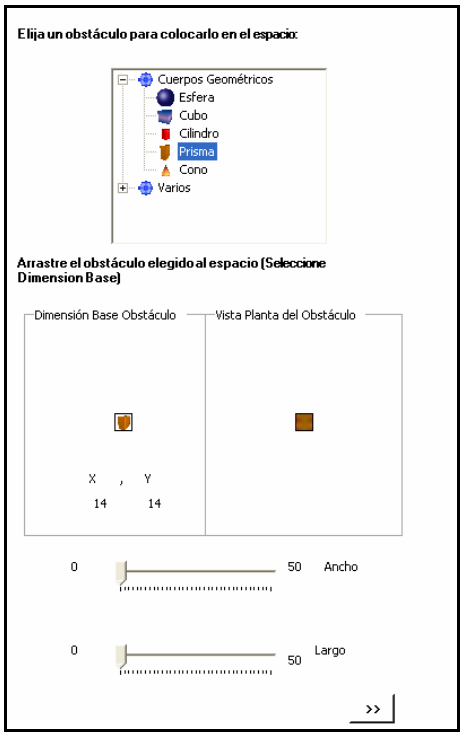


Figura 32. Formulario Obstáculos

FORMULARIO OBJETO MÓVIL

Este formulario contiene una lista de objetos móviles. El usuario debe elegir el tamaño de la base del móvil y luego generar la cuadrícula.

El botón “Dibujar Cuadrícula” activa el método de “descomposición en celdas”, dividiendo el espacio de configuración en una serie de celdas de igual dimensión que el móvil elegido. Así mismo se verifican las celdas libres y las que se encuentran ocupadas por un obstáculo.

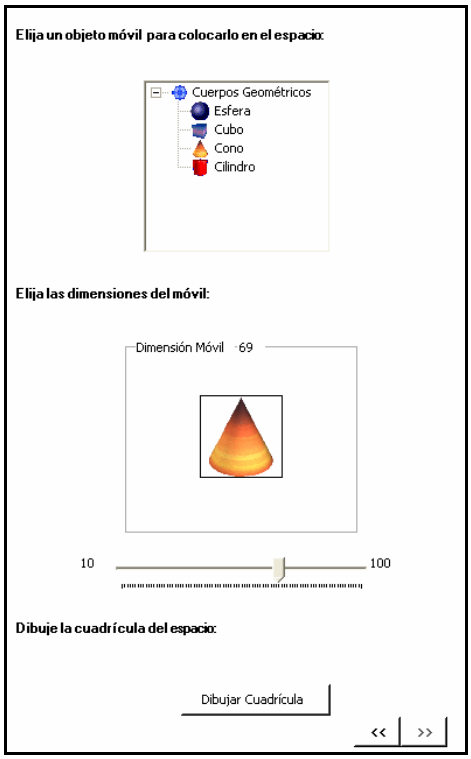


Figura 33. Formulario Objeto Móvil

FORMULARIO COSTOS

Este formulario despliega una lista de materiales y tipos de terreno que pueden ser elegidos por el usuario para cada celda generada por el método de descomposición en celdas. Es de esta forma que el algoritmo considera costos para elegir la ruta óptima.

El usuario elige el tipo de terreno y hace click sobre una celda del entorno para asociar el costo elegido a dicha celda. Gráficamente, los tipos de terreno son representados mediante diferentes colores en el entorno.

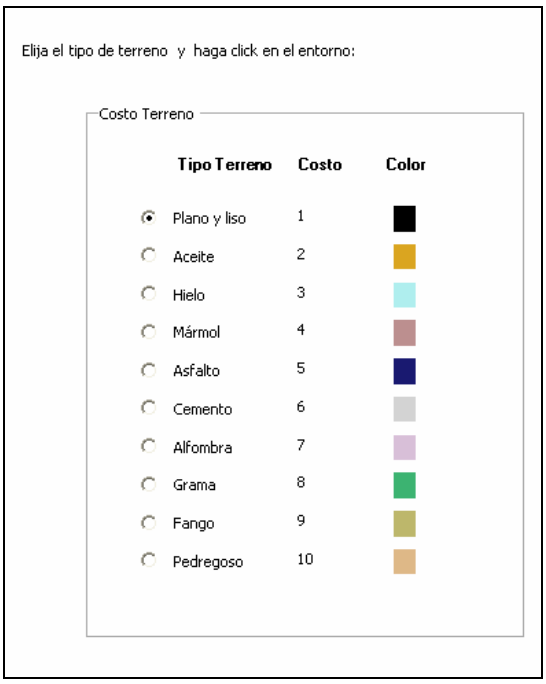


Figura 34. Formulario costos

FORMULARIO POSICIÓN INICIO-DESTINO

El formulario "Posición Inicio-destino" le permite elegir al usuario las posiciones de inicio y destino del objeto móvil, de tal forma que el algoritmo tome como referencias dichas posiciones.

Así mismo, presenta información de la vista "Detalles" que es desplegada al generar la ruta solución. Los datos gráficos que se presentan corresponden a las celdas sucesores, celda actual y celda solución.

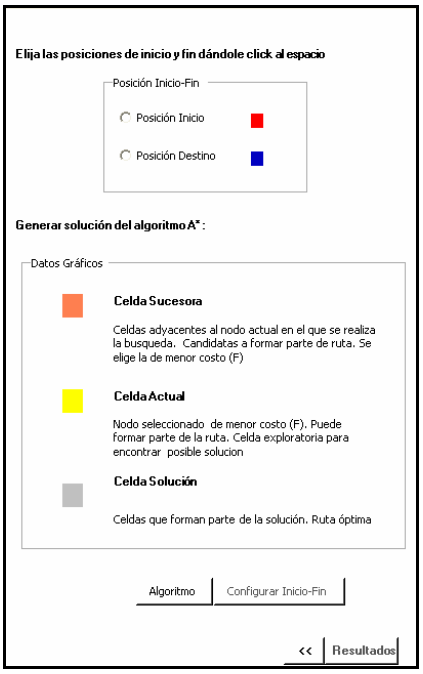


Figura 35. Formulario Posición Inicio-Destino

FORMULARIO SOLUCIÓN

Una vez generada la ruta, el usuario puede visualizar ciertos datos generados por el algoritmo A* que son colocados en el formulario solución.

Los datos son colocados en un componente de Excel que despliega diferentes hojas de acuerdo al tipo de datos que se han generado. Este tipo de componente brinda la ventaja de poder exportar los datos a la aplicación de Excel para luego ser almacenados. Las hojas generadas se detallan a continuación:

1. Sucesores

La hoja sucesores despliega la siguiente información: número de celda consecutivo, número de celda en posición horizontal (x), número de celda en posición vertical (y), celda de la cual es sucesora y los datos de la función evaluadora (f,g,h).

2. Actual

Contiene datos referentes a las celdas que pueden pertenecer a la ruta solución. Dichos datos son: número de celda consecutivo, posición horizontal (X) y posición vertical (Y).

3. Resultado

La hoja “Resultado” muestra información sobre las celdas que forman parte de la ruta solución. La información se detalla a continuación: número de celda consecutivo, posición horizontal (X), posición vertical (Y) y función evaluadora (f,g,h).

4. Estado

Presenta información acerca del estado de una celda, es decir, si esta se encuentra ocupada o no por un obstáculo. La información consiste en: número de celda consecutivo, estado (ocupada o libre), posición horizontal (X) y posición vertical (Y).

5. Obstáculo

Esta hoja presenta información referente a los obstáculos que el usuario ha colocado en el espacio de configuración. La información es la siguiente: número de celda consecutivo, nombre del obstáculo, posición horizontal (X) y posición vertical (Y).

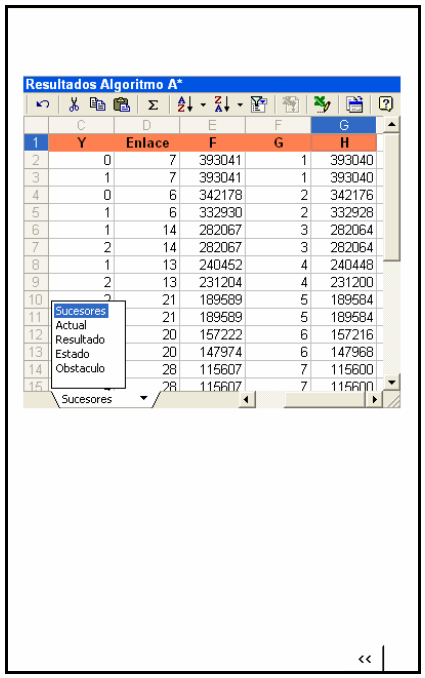


Figura 36. Formulario Solución

FORMULARIO ASISTENTE

Este formulario es desplegado al seleccionar la opción de menú “Asistente Importación/Exportación”. El usuario puede realizar una de estas dos tareas: Guardar grafo ó Importar solución desde un archivo de texto generado por otro algoritmo de búsqueda.

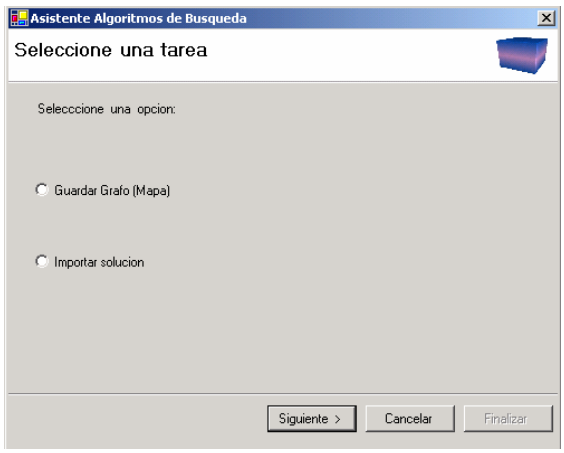


Figura 37. Ventana principal asistente

- **GUARDAR GRAFO**

Esta opción permite guardar la configuración del grafo generado a partir de la ubicación de los objetos dentro del espacio de configuraciones, el estado inicial y el estado final seleccionados por el usuario.

El asistente permite guardar el grafo en dos formatos diferentes:

- Formato conceptual del grafo
  Guarda los vértices, arcos y costo asociados al grafo generado. Además brinda información del estado de cada vértice, es decir, si se encuentra ocupado (1) ó libre (0).

- Formato cuadrícula
  Guarda los vértices del grafo y el estado de cada vértice.

La configuración es almacenada en un archivo de texto según el formato que haya sido seleccionado. De esta forma, una aplicación externa puede tomar dicha información y aplicar un algoritmo de búsqueda para obtener su solución.

Cabe señalar que la solución generada por la aplicación externa debe ser almacenada en otro archivo de texto. La segunda opción del asistente, permite retomar este archivo para mostrar gráficamente la solución dentro de la aplicación.

En la siguiente figura se muestra una ventana del asistente indicando la estructura del grafo cuando se ha seleccionado guardar la configuración en formato conceptual.

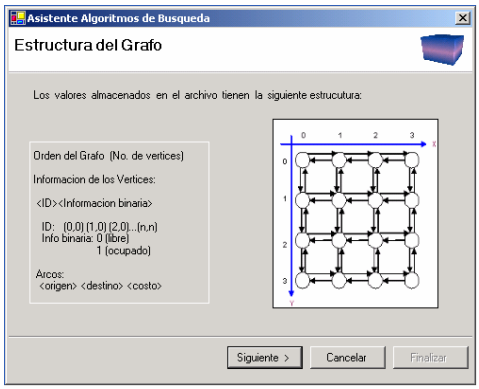


Figura 38. Ventana de asistente de Estructura del Grafo

- **IMPORTAR SOLUCIÓN**

Esta opción permite abrir un archivo de texto con la solución generada por una aplicación externa. La aplicación lee la información y grafica dicha solución.

La siguiente figura muestra la ventana que es desplegada al importar una solución. Permite elegir el archivo de texto generado por una aplicación externa.

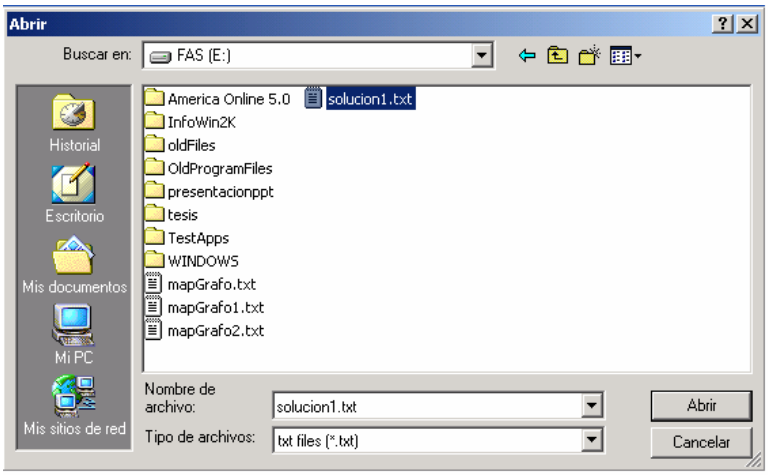


Figura 39. Ventana importación de archivo de texto del asistente

- **FORMATOS DE ARCHIVO UTILIZADOS POR EL ASISTENTE**

Según el tipo de formato del grafo seleccionado por el usuario, en los archivos de texto generados por el asistente se guarda la siguiente información:

a) Numero de vértices o nodos del grafo.

b) Vértices en formato de coordenadas (x,y) o vértices numerados.

c) Información de estado (binaria) de cada vértice 1 (ocupada) o 0 (libre).

d) La posición de inicio y la posición de destino o meta según el formato de los vértices seleccionado.

La siguiente figura muestra la información colocada en el archivo de texto cuando se ha elegido guardar el grafo en formato conceptual. El primer número indica la cantidad de vértices del grafo; las siguientes seis líneas indican la posición X,Y de un vértice y su información binaria. Las líneas restantes muestran los arcos existentes entre cada par de vértices y su costo asociado.

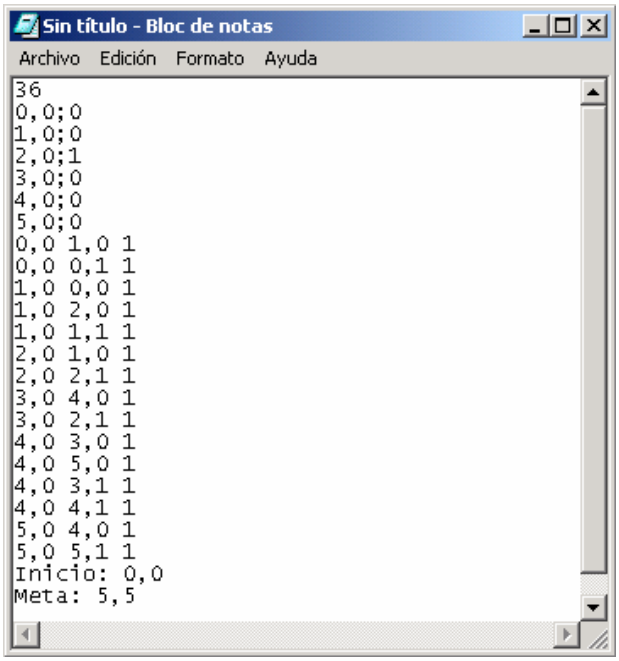


Figura 40 Archivo generado por el asistente según opción formato conceptual

Los archivos de texto importados por el asistente para graficar la solución encontrada por una aplicación externa, deben tener las siguientes características:

a) El valor de los nodos que forma parte de la solución pueden almacenarse de dos formas: como coordenadas (x,y) o nodos numerados.

b) Cada nodo que forma parte de la solución, debe ser almacenado en una línea del archivo, es decir, no deben almacenarse en una sola línea de forma continua.

La siguiente figura muestra la información almacenada en el archivo de texto cuando se ha elegido importar solución durante el asistente.

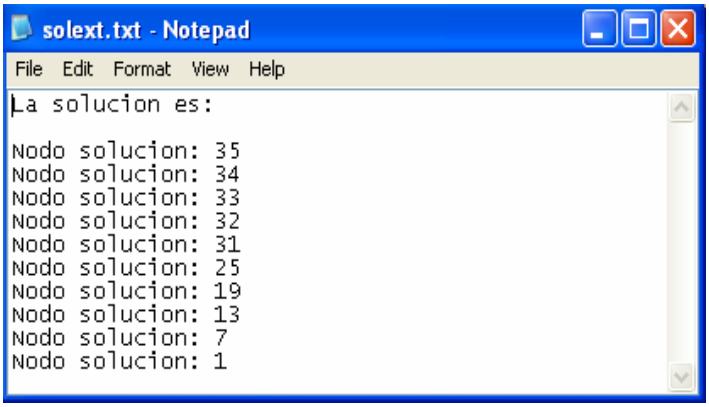


Figura 41 Formato del archivo a ser importado por el asistente

FORMULARIO ESCENA 3D

En este formulario se visualiza el entorno del móvil en tres dimensiones. Se presentan además opciones de configuración del entorno del móvil, de tal forma que el usuario cuente con diferentes formas de visualizar la escena.

Una barra de configuración ha sido colocada en el lado izquierdo del formulario, de tal forma que estos cambios ocurren en tiempo real. Las opciones de configuración incluyen las siguientes características: visualización de la solución, vista de la escena y graficación del terreno en el que se desplaza el móvil. Además, puede encontrarse información de la posición del móvil durante el recorrido de la solución en los ejes X y Z.

El menú correspondiente al presente formulario posee las opciones de configuración de luz de la escena y velocidad de desplazamiento del móvil.

La siguiente figura muestra el formulario “Escena 3D” con sus opciones de configuración y menú principal.

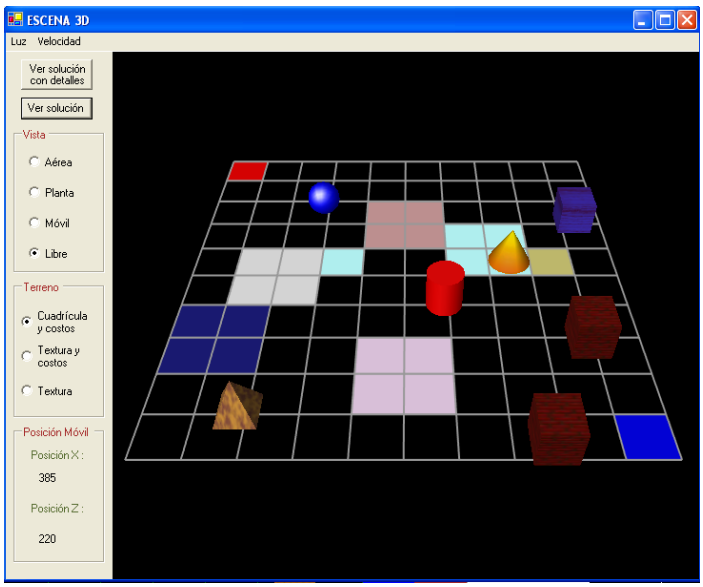


Figura 42. Formulario Escena 3D

3.3 DISEÑO DE MENÚ

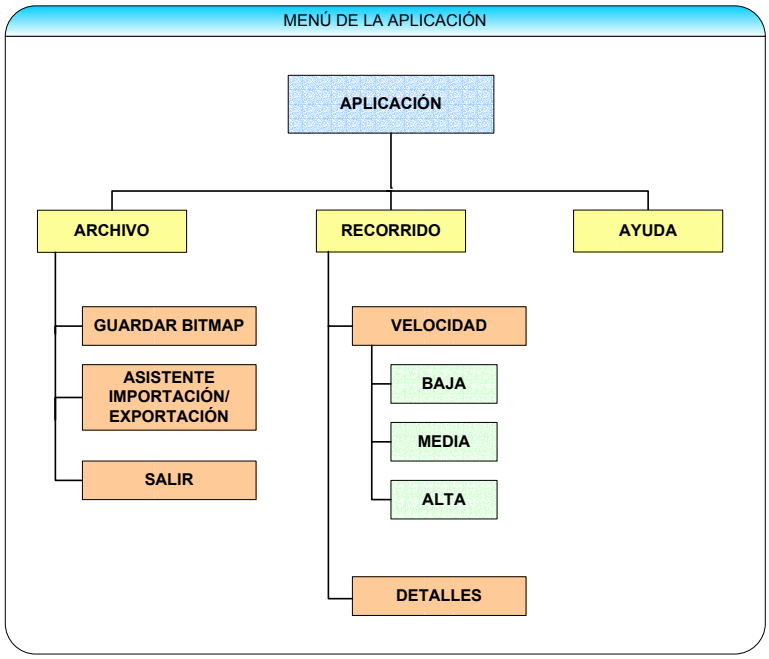


Figura 43. Estructura del Menú

El diagrama anterior muestra las opciones de menú que posee la aplicación. Las opciones principales consisten en la exportación e importación del espacio de configuraciones y opciones de vista del recorrido.

La función de cada una de las opciones se describe a continuación:

ARCHIVO

El menú archivo contiene opciones de importación y exportación de configuraciones del entorno del móvil.

- **Guardar Bitmap**

  Permite guardar el entorno del móvil o espacio de configuraciones como una imagen de tipo bitmap, para ser utilizada como futura referencia.

- **Asistente de Importación/Exportación**

  Permite guardar la configuración del entorno del móvil a un archivo de texto, con el fin de ser utilizado por una aplicación externa que involucre diferentes algoritmos de búsqueda. Así mismo, permite importar la solución generada por dichas aplicaciones.

- **Salir**

  Opción utilizada para cerrar la aplicación.

- **RECORRIDO**

Brinda opciones de visualización de la ruta generada, de tal forma que permite mostrar detalles y regular la velocidad de la vista.

- **VELOCIDAD**

  Permite regular la velocidad en la que se despliega la ruta generada cuando se elige la opción de ejecutar algoritmo. Posee tres opciones: baja, media y alta.

- **DETALLES**

  Despliega detalles generados por el algoritmo de búsqueda A* cuando se genera la ruta. Dichos detalles incluyen: celdas sucesoras, celda actual y celdas solución.

- **AYUDA**

Esta opción brinda información útil para el usuario de la aplicación, brindándole información sobre cómo configurar el entorno del móvil y generar la ruta. Posee información básica de utilización de la aplicación.

4. Conclusiones

- El presente trabajo de investigación brinda las etapas involucradas en la solución del problema de planificación de trayectorias de objetos móviles. La aplicación utiliza el método de descomposición en celdas como método de planificación de trayectorias y al algoritmo A* para realizar la búsqueda de la ruta dentro de un grafo derivado del entorno del móvil.

- Existen varios algoritmos de búsqueda utilizados en el campo de la Inteligencia Artificial para recorrer grafos y encontrar una solución especifica. Estos algoritmos se dividen en búsqueda sin información o búsqueda respaldada con información. Los algoritmos respaldados con información, como en el caso del algoritmo A* resuelven la búsqueda con mayor rapidez debido a que se tienen diferentes criterios como son distancia de un punto a otro, costos de ruta, etc., para generar secuencias de acciones posibles que lleven a un objetivo durante el proceso de búsqueda.

- En esta aplicación se seleccionó el algoritmo A* por poseer la característica de encontrar soluciones óptimas y rápidas, ya que para el problema de la planificación de trayectorias de un objeto móvil es importante encontrar trayectorias con dichas características. La aplicación brinda información obtenida de la ejecución del Algoritmo A* en el problema de búsqueda de trayectorias, de tal forma que pueda ser analizada para una mayor comprensión del funcionamiento del algoritmo.

- Al analizar varias características del algoritmo A* se observó que la solución encontrada corresponde a la mejor ruta para un entorno de configuración específico. Dicha ruta no siempre es la más corta ya que toma en consideración tanto costos como distancia que debe recorrer el móvil.

- La complejidad del algoritmo A* viene dada por la elección de una buena heurística. Su elección depende de las características del problema que intentará solventar el algoritmo. El algoritmo no toma en cuenta los valores heurísticos de otros estados una vez se ha seleccionado un camino que se considera aceptable durante el proceso de búsqueda. Esto incurre en problemas de búsqueda generados por máximos locales, debido a que el algoritmo A* no es capaz de "mirar mas lejos" para encontrar una solución.

- OpenGL incorpora funciones que realizan los cálculos matemáticos involucrados en la construcción de gráficos tridimensionales, de tal forma que facilita y agiliza la programación de este tipo de escenarios sin tener un conocimiento amplio de álgebra lineal.

- Por medio de la simulación en tres dimensiones del recorrido de objetos móviles, se puede llegar a predecir con mayor exactitud su comportamiento en un escenario real. En general, la simulación en tres dimensiones agrega mayor realismo a diversas aplicaciones en los campos científico, médico, industrial, técnico, educativo y entretenimiento.

- La aplicación desarrollada puede ser utilizada como herramienta didáctica, a fin de analizar y comparar la diversidad de algoritmos de búsqueda que existen en el campo de la Inteligencia Artificial. Y más aún ser utilizada como referencia para implementar una aplicación de simulación de robots.

5. Recomendaciones

- Agregar figuras más complejas a las listas de obstáculos y objetos móviles con el fin de obtener mayor realismo en la simulación.

- Permitir cambiar las texturas de los obstáculos y del móvil de acuerdo a un archivo de imagen brindado por el usuario.

- Incluir configuraciones adicionales a la escena en tres dimensiones tales como color de fondo, niebla, sombra, etc.

- Modificar la aplicación, de tal forma que el objeto móvil tenga la capacidad de aprender rutas ya recorridas.

- Estudiar la posibilidad de incluir dentro de la aplicación, algoritmos adicionales que resuelvan el problema de planificación de trayectorias de un objeto móvil con el fin de compararlos.

- Permitir guardar la configuración del entorno del móvil en un archivo, de tal forma que pueda ser cargada posteriormente.

- Añadir y modificar propiedades físicas del objeto móvil, de tal forma que se asemejen a las de un robot.

BIBLIOGRAFÍA

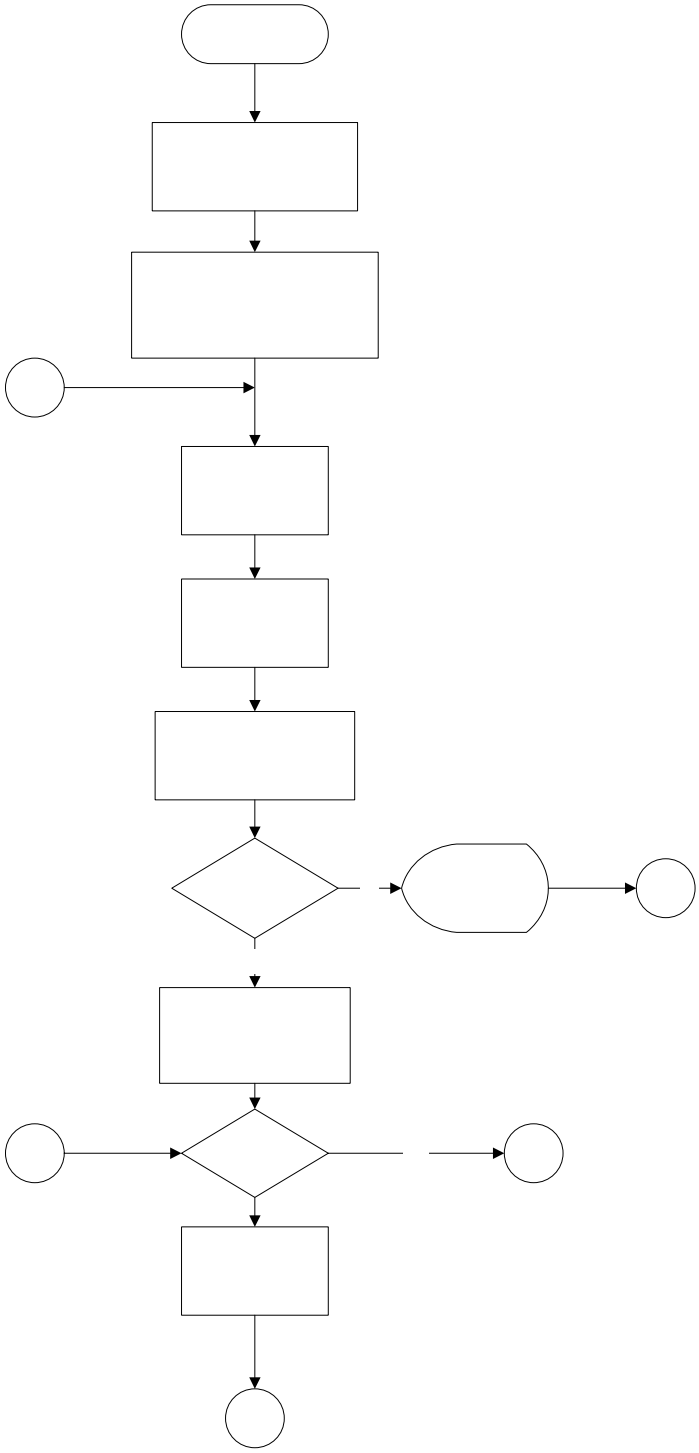
- Adams, Lee (1986). “High-Performance”. CAD graphics in C. Windcrest/McGraw-Hill.
- Borja Ernesto, Ghazarian Anthony, Salazar Rodolfo, Santos Ricardo (1992). “Construcción de un robot autoprogramable: introducción a la robótica”. Tesis ingeniería. Universidad Centroamericana José Simeón Cañas (UCA).
- Cabezas, Horacio (2000). “Metodología de la Investigación”. Editorial Piedra Santa.
- Diccionario de la Lengua Española (1992). Real Academia española.
- Heileman, Gregory L. (1998). “Estructuras de Datos, Algoritmos, y Programación Orientada a Objetos”. McGraw Hill.
- Hernández Sampieri Roberto, Fernández Collado Carlos, Baptista Lucio Pilar (1991). “Metodología de la Investigación”. McGraw Hill.
- Joyanes Luis, Zahonero Ignacio (1998).” Estructura de Datos”. Algoritmos, abstracción y objetos. McGraw Hill.
- Kendall & Kendall (1997). “Análisis y Diseño de Sistemas”. Tercera edición. Prentice Hall.
- NEC Research Institute ResearchIndex
  http://citeseer.nj.nec.com/
- Rich Elaine, Knight Kevin (1994). “Inteligencia Artificial”. MC Graw Hill.
- Russel Stuart, Norving Peter (1996). “Inteligencia Artificial” Un enfoque moderno. Prentice Hall.
- Winston Patrick Henry (1994). “Inteligencia Artificial”. Tercera edición. Addison-Wesley Iberoamericana.
- Wreight Richard, Sweer Michael (2000).”OpenGL SuperBible”. Second Edition. Waite Group Press.

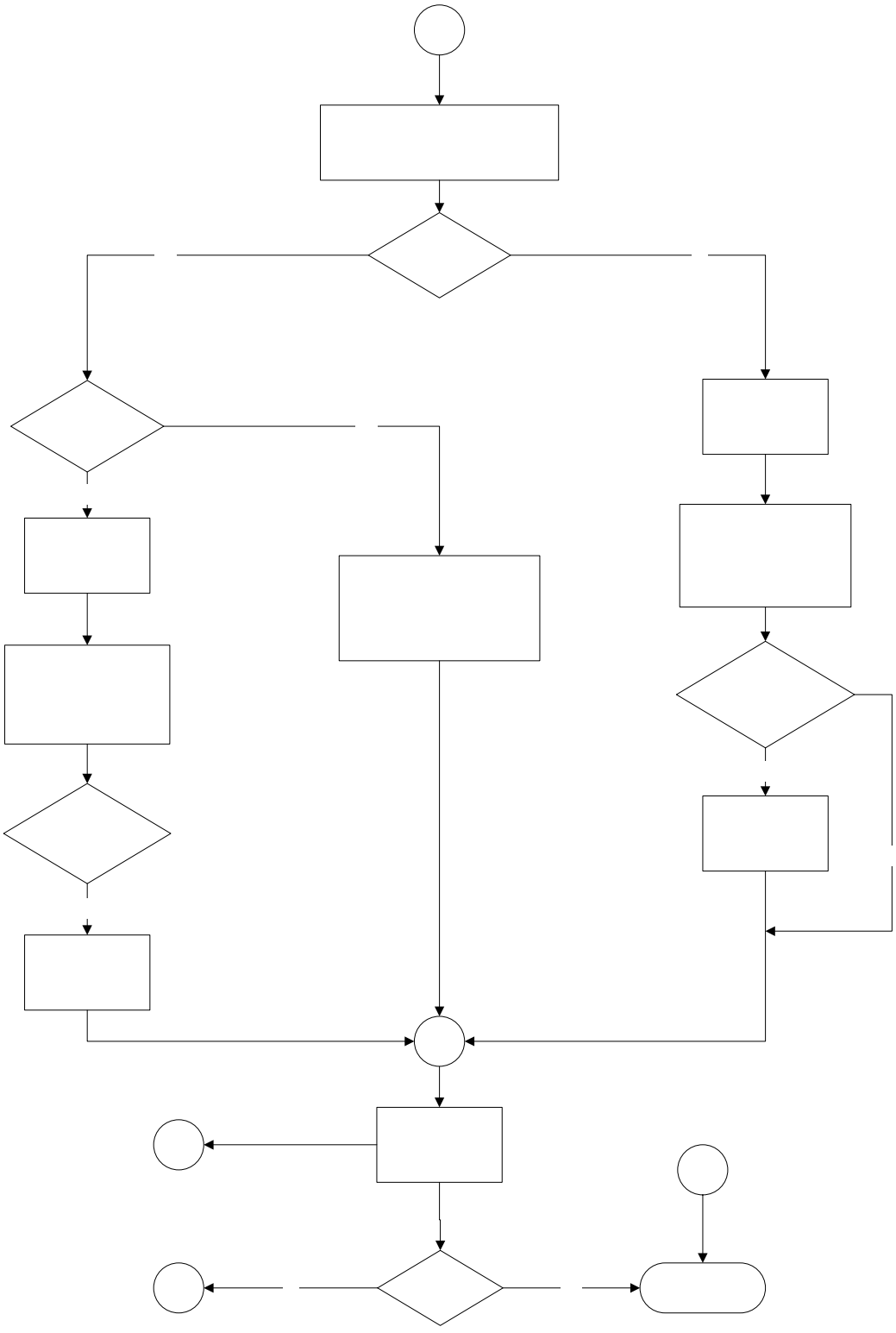
GLOSARIO

- Algoritmo: especificación concisa de un método para resolver un problema.
- Búsqueda con información: es aquella en la que se utiliza conocimiento específico para un problema determinado, permiten encontrar soluciones con más eficiencia que una búsqueda sin información.
- Búsqueda sin información: en este tipo de búsqueda, no existe información acerca de la cantidad de pasos necesarios para o sobre el costo de ruta para pasar del estado de un momento dado a la meta. Lo único que permiten hacer es diferenciar entre un estado meta del otro que no lo es.
- Clase: abstracción que representa a un conjunto de objetos con un comportamiento o interfaz común. Consta de métodos y datos que resumen las características comunes de un conjunto de objetos.
- Grafo: Consiste en un conjunto de vértices o nodos V y un conjunto de arcos A. Se representa con el par G=(V,A).
- Inteligencia Artificial: campo de estudio que se enfoca a la explicación y emulación de la conducta inteligente en función de procesos computacionales.
- Lista: secuencia de cero o más elementos de un mismo tipo. Este tipo de objeto abstracto se representa <e1,e2,…,en>.
- OpenGL: interfaz de software para manipulación de hardware de gráficos.
- Tercera dimensión (3D): gráficos tridimensionales que constan de ancho, alto y profundidad. En gráficos por computadora, las imágenes en tercera dimensión son descritas por las coordenadas x,y,z.
- Trayectoria: línea descrita en el espacio por un punto que se mueve.

ANEXOS

Anexo No.1 Diagrama de flujo del Algoritmo A*





No

Anexo No.2 Características del Algoritmo A*

En general, para demostrar como se puede llevar a la práctica un algoritmo de búsqueda para el problema de planificación de trayectorias de un objeto móvil, se deben tomar en cuenta los siguientes elementos:

a) Un grafo de conectividad creado a partir de un método de planificación de trayectorias.
b) La posición de inicio (nodo de inicio del grafo) y la posición de destino (objetivo o meta) del objeto móvil.
c) Diseño de un algoritmo de búsqueda sobre grafos que permita encontrar una ruta por donde se desplazara el objeto móvil.

Con respecto a los algoritmos de búsqueda sobre grafos se pueden enumerar las siguientes características:

a) Durante el proceso de búsqueda cada nodo del grafo representa un estado del problema, y cada arista entre un nodo y otro representa una relación entre los estados.
b) El proceso de búsqueda debe encontrar uno o más caminos que conecten un estado inicial (nodo de inicio) con un estado final (nodo objetivo o meta).
c) La dirección en la que se va a conducir la búsqueda se puede realizar buscando hacia delante, es decir, partiendo del nodo inicial del grafo hacia el nodo objetivo, o se busca hacia atrás, partiendo del objetivo.
d) Cada nodo del grafo se expande generando un conjunto de sucesores según las reglas (operadores) que definen los movimientos permitidos en el espacio del problema. En la aplicación se ha tomado como regla expandir aquellos sucesores que se encuentren adyacentes a un nodo y libres de obstáculos

según una información binaria almacenada en cada nodo del grafo indicando si esta ocupado (1) o libre (0) de obstáculos.

e) Por lo general se utilizan estructuras de datos de tipo lista para almacenar los valores de los nodos y realizar el recorrido del grafo durante el proceso de búsqueda.
f) Durante el proceso de búsqueda se puede disponer o no de información. Dicha información se encuentra asociada a cada nodo del grafo y aristas, por ejemplo: el costo de moverse de un nodo a otro, distancia entre un nodo y el objetivo, etc.

Tomando como referencia las características descritas anteriormente, se enumeran a continuación los componentes que forman parte del algoritmo A*:

1. Función evaluadora: *f = g + h*

El algoritmo A* toma en cuenta la estimación (heurística = h) de la distancia entre un nodo y el objetivo, y el coste (g) del camino encontrado desde el nodo de inicio hasta el objetivo.

Con la incorporación de *h* en *f* la búsqueda estará dirigida por cierto conocimiento específico, aunque no totalmente seguro, pero no se garantiza el hallazgo de una solución óptima (expansión del menor número de nodos durante el proceso de búsqueda).

Con la incorporación de *g* en *f* se garantiza ir calculando el coste de los nodos del grafo que van siendo expandidos, eligiendo en cada paso el de menor costo.

El algoritmo A* combina *g* y *h* por medio de la suma de estos valores dentro de la función *f*, esto implica que el algoritmo A* puede usarse si se esta interesado en encontrar un camino de coste total mínimo o simplemente el camino mas rápido posible.

2. Listas utilizadas por el algoritmo A*

Las listas utilizadas por el algoritmo A* se denominan: lista ABIERTA y lista CERRADA.

La *lista abierta* almacena todos los nodos que se han generado y a los que se les ha aplicado la función de evaluación, pero todavía no se han expandido sus sucesores. Los nodos de esta lista son ordenados según el menor valor de *f*.

La *lista cerrada* almacena todos los nodos que ya fueron seleccionados para su expansión (según el menor valor de *f* seleccionado de la lista abierta).

3. Análisis

Con el objetivo de analizar el comportamiento del algoritmo A* se detallan los siguientes ejemplos:

3.1 Ejemplo 1

Consideremos el espacio de estados que se muestra en la figura 32 en el que los valores heurísticos de los estados (nodos del grafo) están escritos entre paréntesis, con estado inicial I y con un único estado objetivo O.

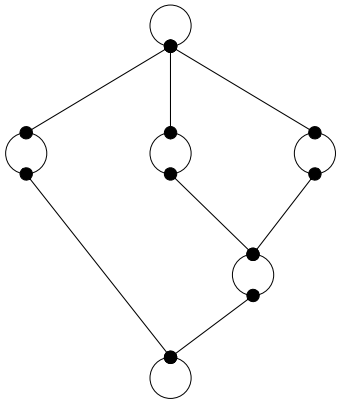


Figura 44

La solución encontrada por el algoritmo para el grafo de búsqueda anterior se muestra en la figura 33. Los números a continuación de los nodos del grafo corresponden a los valores de *g* y *h*, respectivamente.

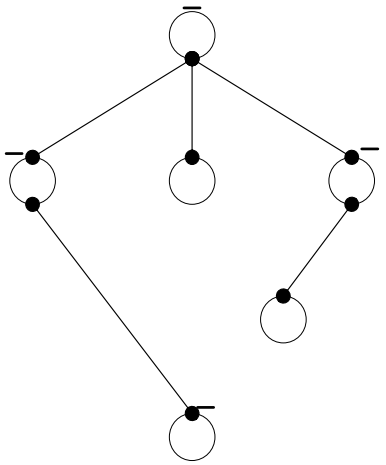


Figura 45

Haciendo énfasis en la solución de la figura anterior, a continuación se describe el proceso de selección y almacenamiento de los nodos del grafo dentro de las listas utilizadas por el algoritmo:

- El nodo inicial (I) es almacenado en la lista ABIERTA. Al iniciar la búsqueda el único elemento dentro de esta lista es I, por lo que es este el primer nodo que se expande.

- Al expandir I (**1**) se generan los nodos sucesores A, B y C almacenándose en la lista ABIERTA. Los valores correspondientes de ***f*** (g + h) para cada nodo sucesor son: Nodo A: 6 (3+3); Nodo B: 7 (2+5); Nodo C: 3 (1+2).

- Los nodos en la lista ABIERTA son ordenados con respecto al menor valor de ***f*** quedando ordenados de la siguiente forma: C, A y B.

- La ordenación respecto a ***f*** en la lista ABIERTA permite que se seleccione el nodo C para expandirlo.

- Al expandir C (**2**) se genera el nodo sucesor D. El nodo D pasa a formar parte de la lista ABIERTA y C a la lista CERRADA puesto que ya fue seleccionado. Cabe señalar que para cada nodo dentro de la lista CERRADA se guarda información de su nodo padre o antecesor.

- Hasta este momento los valores de ***f*** para cada nodo dentro de la lista ABIERTA son: Nodo A: 6; Nodo B: 7; Nodo D: 7 (4 + 3).

- La ordenación respecto a ***f*** en la lista ABIERTA permite que se seleccione el nodo A para expandirlo.

- Al expandir A (**3**) se genera el nodo sucesor O. El nodo O pasa a formar parte de la lista ABIERTA y A ha la lista CERRADA.

- El nodo O dentro de la lista ABIERTA tiene un valor de *g* igual a 6 y un valor de *h* igual a 0. Cabe señalar que la búsqueda termina hasta que el nodo objetivo es seleccionado para su expansión.

- La ordenación respecto a ***f*** en la lista ABIERTA permite que se seleccione el nodo O (**4**) para expandirlo. En este momento el algoritmo detecta que O es el nodo objetivo o meta y la búsqueda finaliza devolviendo la solución. La solución se calcula recorriendo la lista CERRADA según la información de los antecesores o padre de cada nodo a partir del objetivo (O), que en este caso son el nodo O, A e I (pasos 4, 3 y 1 respectivamente).

3.2 Ejemplo 2

Es evidente que en el ejemplo anterior, el algoritmo A* ha encontrado un camino solución optimo, y de menor longitud (2), pero, en general, no siempre presenta una ruta con estas características.

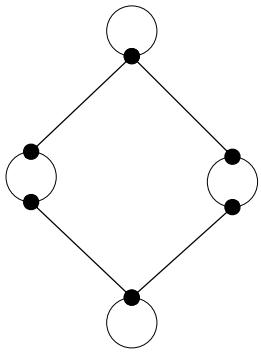


Figura 46

Con el grafo de búsqueda de la figura 34 se obtiene una solución de la siguiente manera:

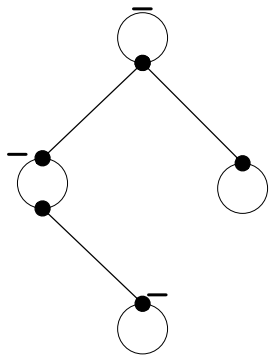


Figura 47

Puede observarse que la ruta solución encontrada por el algoritmo no es la óptima. La razón por la que en este segundo ejemplo la solución óptima no es encontrada es por el valor heurístico de B (h=4). El valor de *h* es demasiado grande (se puede ir hasta O con coste 2). Al ser este valor muy grande hace que la ordenación respecto a ***f*** (g+h) en la lista ABIERTA resulte engañosa y el nodo A pase por delante del nodo B, perdiéndose el camino óptimo.

Para generalizar este análisis, se detallan a continuación algunas nociones teóricas:

Un programa de búsqueda puede encontrarse con un máximo local. Un *máximo local* es un estado que es mejor que todos sus vecinos, pero que no es mejor que otros estados de otros lugares. En un máximo local todos los movimientos producen estados peores. Los máximos locales son particularmente frustrantes porque frecuentemente aparecen en las cercanías de una solución.

Observando el espacio de estados de la figura en mención, puede apreciarse que el nodo A es un máximo local, ya que en un momento de la búsqueda es mejor que el nodo B (vecino), pero como se encuentra cerca de la solución ya no es posible generar nuevos estados mejores que el.

Para evitar un problema de máximo local, asumiendo que el proceso de búsqueda se realiza en un espacio de estados grande, se puede volver atrás hacia algún nodo anterior e intentar seguir un camino diferente. Es especialmente razonable si el nodo posee otra dirección que de la impresión de ser tan o casi prometedora como la que se eligió. Para implementar esta estrategia, se debe mantener una lista de caminos que casi se han seguido y volver a uno de ellos si el camino actual que se ha seguido da la impresión que no hay salida o se encuentra obstaculizado.

En general, puede concluirse que esta situación de máximo local puede darse debido a que el algoritmo A* no es capaz de “mirar mas lejos” para encontrar una solución, pero también se puede evitar tratando de modificar la función heurística, sobre todo cuando el valor de *h* cambia bruscamente al alejarse de una solución.

3.3 Ejemplo 3

Con respecto al papel que desempeña la función *g*, se puede decir que no siempre se elige como el siguiente nodo a expandir aquel que parece más cercano al objetivo (medido por h). Este valor es útil si el camino elegido tiene relevancia (de menor costo). Si solo es relevante alcanzar el objetivo sea de la forma que sea, se puede definir *g* como 0, de forma que siempre se elegirá el nodo que parezca mas cercano al objetivo final.

Si lo que se desea es encontrar un camino con el menor número de pasos posibles (de menor longitud) se hace que el costo aplicado para pasar de un nodo a otro sea 1. Esta es una característica de la búsqueda primero en achura, que a pesar que no dispone de información durante el proceso de búsqueda siempre encuentra una ruta de menor longitud. A continuación se compara la búsqueda en anchura con el algoritmo A* utilizando el mismo espacio de estados de la figura 32.

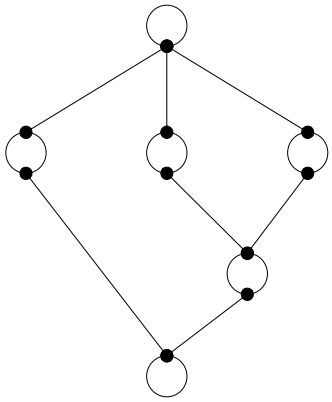


Figura 48

La solución encontrada por el algoritmo A* para el grafo de búsqueda anterior se muestra en la figura 37. Los números a continuación de los nodos del grafo corresponden a los valores de g y h, respectivamente.

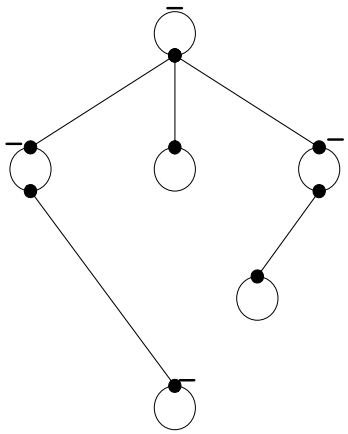


Figura 49

Es de hacer notar que en este caso los nodos expandidos (4) son los mismos que cuando se consideran costos, aunque no quiere decir que siempre sea así. El valor de f para el nodo D es igual a 5 que es mayor al del nodo A (f = 4), por lo tanto el nodo a expandir dentro de la lista ABIERTA vuelve a ser A, encontrándose el nodo sucesor O que es el objetivo. El camino solución vuelve a ser el nodo O, A e I. Es evidente que en este caso el algoritmo A* encontró el camino mas corto al considerar un costo constante (g = 1).

En la siguiente figura se encuentra el mismo espacio de estados resuelto por el algoritmo de búsqueda en anchura.

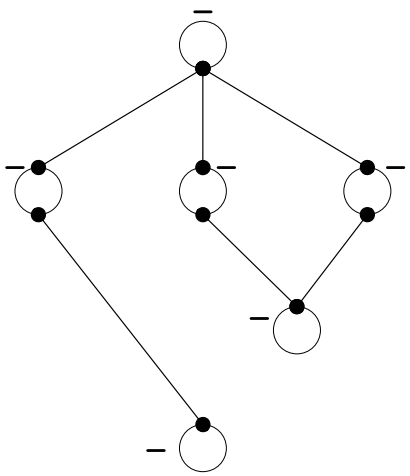


Figura 50

Es importante destacar que el algoritmo de búsqueda en anchura recorre el grafo por niveles, es decir, según el nivel de profundidad del grafo. Cada nodo sucesor es agregado al final de la lista ABIERTA, así se consigue que los nodos en la lista ABIERTA estén ordenados según su nivel de profundidad (en orden decreciente).

Es de hacer notar que este algoritmo al no disponer de información, los nodos de igual profundidad generados a partir de un mismo padre se ordenan de un modo arbitrario, que para el caso de la solución de la figura anterior los nodos de igual nivel se han expandido de "derecha a izquierda" (Nodo C, B y A respectivamente). Además el algoritmo toma en cuenta que mientras no se hayan expandido todos los nodos de un nivel, no pueden seleccionarse nodos en niveles más profundos.

Cabe señalar que el recorrido en anchura examina todos los estados que son accesibles desde el estado inicial. Esto quiere decir que el estado objetivo aparecerá en un cierto nivel de profundidad y, en un número finito de pasos

llegara a ese nivel. Este razonamiento muestra que el algoritmo de búsqueda en anchura siempre encuentra la secuencia solución más corta (una de las que tiene el mínimo número de aristas). Sin embargo, comparando con la solución encontrada por el algoritmo A* se puede deducir que durante el proceso de búsqueda, el algoritmo A* ha expandido un menor numero de nodos (4) que el de búsqueda en anchura (5).

Para finalizar con el análisis de g, se puede decir que si se desea encontrar un camino de menor coste de forma que algunos operadores tengan mayor coste que otros, se hace que el coste aplicado al pasar de unos nodos a otros refleje estos costes. De esta forma el algoritmo A* puede usarse si se esta interesado en encontrar un camino de coste total mínimo.

3.4 Ejemplo 4

Analizando el papel que desempeña la función *h* (informa de la distancia de un nodo al objetivo), se puede decir que si el valor de *h* de un nodo sucesor (*h'*) hace una estimación perfecta de *h*, entonces el algoritmo A* converge inmediatamente hacia el objetivo. Si, por otro lado, el valor de *h'* es siempre 0, la función *g* será la que controle la búsqueda.

Es importante destacar que durante el proceso de búsqueda nunca se verifica que *h* de un nodo seleccionado sea menor o igual que *h* de un nodo sucesor. Generalizando si 'e' es un estado desde el que podemos encontrar algún camino hasta alguno de los estados objetivo, se puede definir:

h*(e) = un camino mínimo de e hasta alguno de los estados objetivo.

Por lo tanto *h* y *h** no son comparables, es decir, no se verifica que h(e) <= h*(e) para todo e, ni tampoco que h(e) >= h*(e) para todo e. Esto significa que los

nodos dentro de la función ***f*** son ordenados con respecto al valor de *g* más no de *h*.

Retomando el ejemplo analizado en las figuras 34 y 35, se pueden reinterpretar los números que ahí aparecen diciendo que *h* del nodo B (4) es mayor que *h** del nodo B (3), lo que implica que el valor de ***f*** (g+h) para el nodo B (3+4) es mayor que el de ***f**** igual a 5 (3+2). De hecho, el valor de *f* para el nodo B es también mayor que el valor de *f* para el nodo objetivo O (6 + 0). Esto implica que al ordenar la lista ABIERTA, este ultimo nodo pasa por delante del nodo B incorrectamente, puesto que ***f**** es menor que ***f*** del nodo que contiene en ese momento al nodo O.

La situación anterior no podrá producirse si *h* es una subestimación de *h**. Si la heurística *h* verifica que h(e) <= h*(e) para todo estado e (para el que exista h*) se dice que *h* es una *heurística admisible*. La importancia de esta noción radica en que si el algoritmo A* utiliza una heurística admisible, entonces, si existe solución, encuentra una solución optima y seguramente de menor longitud.

Cabe señalar que si se toma como heurística la función idénticamente nula (h=0), siempre subestima h*, es decir, siempre es admisible y, por tanto el algoritmo A* siempre encontrara una solución optima. Pero si h es igual a 0, entonces *f* = *g*; es decir, la búsqueda estará controlada totalmente por la información de *g* y el algoritmo se comportara como una búsqueda “primero el menos costoso” o de costos mínimos.

Evidentemente la admisibilidad de una heurística tiene importantes repercusiones en el comportamiento del algoritmo A*. Además el calculo de *h**(e) no es sencillo de implementar (por ejemplo una solución al problema de máximos locales).

Anexo No.3 Resultados

A lo largo del presente documento se han descrito los componentes básicos de la aplicación: método de descomposición en celdas, generación del grafo de conectividad, algoritmo A* y graficación de la trayectoria. Este apartado consiste en la descripción de las capacidades y uso de los componentes anteriormente detallados, uso de la aplicación y resultados obtenidos.

Los resultados incluidos constituyen un análisis al algoritmo A*, presentando sus habilidades, capacidades y limitaciones, de tal forma que futuros trabajos enriquezcan el trabajo realizado.

Los resultados presentados corresponden a cuatro pruebas desarrolladas en las que se realiza el mismo problema de trayectoria en la aplicación. Durante los ejemplos presentados a continuación se consideraron los siguientes factores con el fin de comparar la incidencia de los factores variables en la ejecución del algoritmo.

Factores constantes:

- Número de obstáculos: 14
- Tipo de obstáculos: Esferas
- Dimensión base de obstáculos: 10 píxeles de diámetro
- Dimensión del móvil: 10 píxeles

Factores variables:

- Heurística utilizada en la ejecución del algoritmo
- Costo asociado a las celdas del entorno

PRUEBA 1

El algoritmo A* combina las ventajas que ofrecen los algoritmos: búsqueda avara que permite reducir al mínimo el costo de la meta (h) y la búsqueda por costo uniforme, que reduce al mínimo el costo de la ruta(g); de tal forma que la función de evaluación resultante es: $f(n)= g(n) + h(n)$.

Las pruebas realizadas a lo largo del presente apartado evaluarán los dos componentes de la función evaluadora del algoritmo A*.

- **Especificaciones**

Con fines de evaluación, la primera prueba consiste en crear una ruta cuyo costo (g) es constante e igual a uno, indicando que el desplazamiento se realizará sobre un terreno plano y liso.

El segundo factor a tomar en cuenta es la heurística utilizada (h). Durante esta prueba se ha considerado la siguiente fórmula como heurística: $h^2 = (X_2-X_1)^2 + (Y_2-Y_1)^2$.

Para el planteamiento de la heurística se han considerado dos puntos, $A(X_1,Y_1)$ y $B(X_2,Y_2)$ en el sistema de coordenadas rectangulares. Cada punto corresponde a la esquina superior izquierda de una celda dentro del espacio de configuración. Para verificar la fórmula heurística se puede estudiar la siguiente figura:

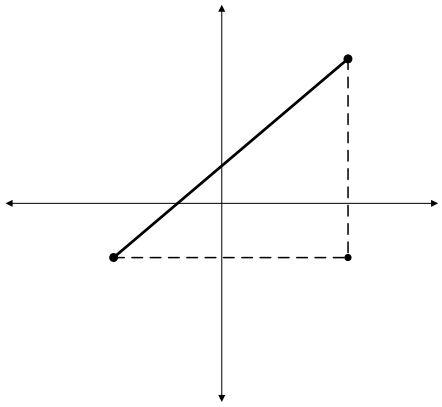


Figura 51 . Diagrama coordenadas

Como AB es la hipotenusa del triángulo rectángulo ABC, el teorema de Pitágoras señala: $AB^2 = AC^2 + CB^2$. Pero: $AC= X_2 – X_1$ y $CB = Y_2 – Y_1$. Por consiguiente: $h = AB^2 = (X_2-X_1)^2 + (Y_2-Y_1)^2$.

- **Ejecución**

La posición de inicio fue ubicada en la celda 1589 y la posición destino en la celda 1349. Se mantuvo la configuración según los factores constantes declarados anteriormente.

La disposición de los obstáculos y las posiciones de inicio y destino pueden ser observadas en la siguiente figura:

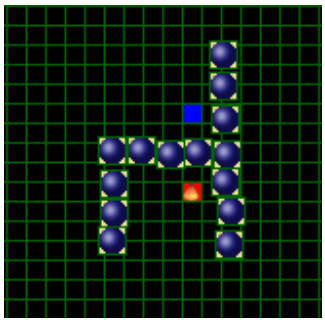


Figura 52. Espacio de configuración prueba 1

La ejecución del algoritmo A* generó los siguientes resultados:

- Posición Inicio en celda 1589
- Posición destino en celda 1349
- Número de celdas sucesoras generadas: 93
- Número de celdas actuales generadas: 63
- Número de celdas que forman parte del resultado: 31 (tomando en cuenta posición de inicio y destino)

La imagen generada por la aplicación que muestra detalles (celdas sucesoras, actual y resultado) de la ejecución del algoritmo se presenta a continuación:

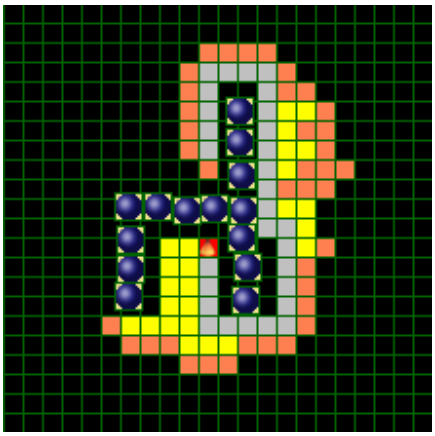


Figura 53. Ruta solución prueba 1

Si se analizan los primeros datos generados por el algoritmo, se podrá observar que dos celdas sucesoras corresponden a la posición de inicio, que es tomada en un primer momento como la celda actual. El segmento de grafo correspondiente a la celda actual se puede observar a continuación:

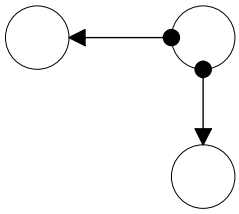


Figura 54. Segmento grafo conectividad prueba 1

Una vez generadas las celdas sucesoras, el algoritmo elige a la celda sucesora con menor resultado en la función evaluadora *F* para convertirse en el siguiente nodo ha ser explorado o celda actual. En este caso, elige a la celda 1588 cuyo valor en *F* es 18, resultado menor al de la celda sucesora 1649, cuyo valor en *F*

es de 26. *Los* datos generados para las celdas sucesoras de la posición de inicio se detallan en la siguiente tabla:

Celda	X	Y	Enlace	F	G	H
1588	28	26	1589	18	1	17
1649	29	27	1589	26	1	25

Tabla 3. Celdas sucesoras de posición inicio en prueba 1

Posteriormente, expande las celdas sucesoras de 1588 y repite el mismo proceso hasta encontrar la posición destino.

- **Observaciones**

El algoritmo A* bajo las condiciones presentadas anteriormente logra encontrar la ruta entre la posición inicio y destino, de tal forma que expande los nodos por orden de *F* creciente.

Sin embargo, cabe señalar que la ruta generada no corresponde a la ruta más corta. Tomando en consideración que los costos son constantes para cada una de las celdas que conforman el espacio de configuración, el algoritmo A* toma como factor principal la heurística (h).

PRUEBA 2

- **Especificaciones**

La segunda prueba consiste en crear una ruta en la que el factor costo (g) es tomado en cuenta asignando costos diferentes a las celdas. En las observaciones de la prueba anterior se hizo notar que la ruta más corta no fue encontrada, es por ello, que se asignará un costo menor a las celdas que podrían formar parte de la ruta más corta, y un costo mayor a las celdas que forman parte de la ruta solución de la prueba anterior.

En esta prueba, la heurística (h) será igual a la anterior, siendo: $h^2 = (X_2-X_1)^2 + (Y_2-Y_1)^2$.

- **Ejecución**

La ruta solución generada en la prueba anterior involucra el desplazamiento del móvil a través de 31 celdas, de la que se concluye que no es la más corta, ya que puede existir otra ruta que se desplaza por 23 celdas. Esto es posible visualizarlo en la siguiente figura:

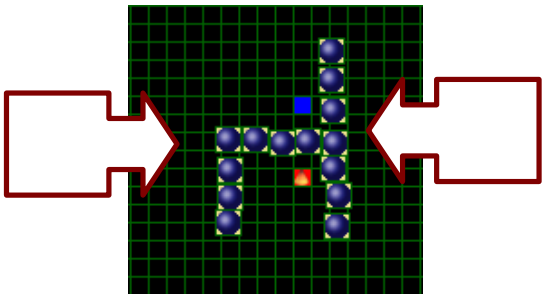


Figura 55. Espacio de configuración prueba 2

Debido a ello, se ha elegido asignarle un costo diferente a las dos posibles rutas para comprobar la incidencia de costos dentro del algoritmo A*. Se han elegido diferentes materiales o tipos de terreno para las dos rutas: *hielo* cuyo costo es *tres* para la ruta más corta y *asfalto* cuyo costo es de *cuatro* para la ruta generada en la prueba anterior.

La posición de inicio corresponde siempre a la celda 1589 y la posición destino a la celda 1349. La configuración del entorno se puede observar en la siguiente figura:

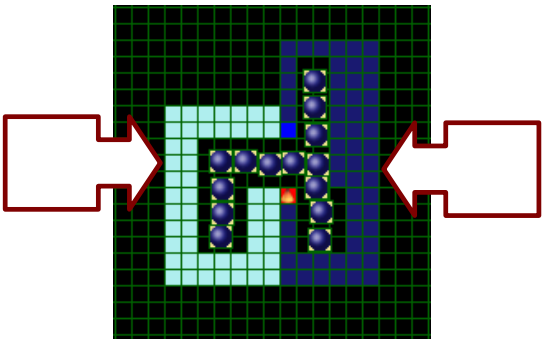


Figura 56. Espacio de configuración con costos prueba 2

La ejecución del algoritmo A* generó los siguientes resultados:

- Posición Inicio en celda 1589
- Posición destino en celda 1349
- Número de celdas sucesoras generadas: 93
- Número de celdas actuales generadas: 85
- Número de celdas que forman parte del resultado: 25 (tomando en cuenta posición de inicio y destino)

La imagen resultado se muestra a continuación:

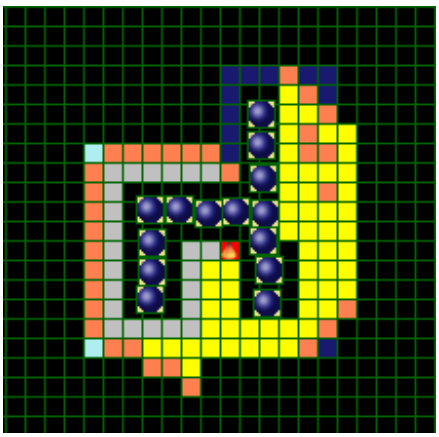


Figura 57. Ruta solución prueba 2

El primer nodo actual que toma es el de la posición de inicio y genera dos sucesores, cuya información se presenta en la siguiente figura:

Celda	X	Y	Enlace	F	G	H
1588	28	26	1589	20	3	17
1649	29	27	1589	30	5	25

Tabla 4. Resultados celdas sucesoras de celda origen en prueba 2

Hay que hacer notar la diferencia en valores de *F* entre las celdas sucesoras de esta prueba y la anterior, debido a que costos de terreno son incluidos. La celda 1588 posee un costo *G* de tres correspondiente a hielo y la celda 1649 posee un costo de asfalto.

- **Observaciones**

El algoritmo A* bajo las condiciones de esta prueba logra encontrar la ruta más corta entre la posición inicio y destino. El factor principal para esta prueba son los costos elegidos para las dos posibles rutas.

Puede observarse cómo el algoritmo A* toma en consideración tanto a la heurística como a los costos, debido a que paralelamente va expandiendo nodos del lado derecho e izquierda de la posición de inicio. Sin embargo, los valores de *F* para la ruta solución de la prueba anterior son mayores que los obtenidos en la solución brindada por el algoritmo para esta prueba, de tal forma, que no expande más nodos del lado derecho y sigue expandiendo nodos del lado izquierdo. Los valores para *F* del lado izquierdo son menores debido al menor valor de costo asignado .

En esta prueba puede notarse la incidencia de los costos para la obtención de la ruta más corta y cómo el algoritmo A* valora tanto costos como la heurística empleada.

PRUEBA 3

- **Especificaciones**

Esta prueba consiste en crear una ruta cuyo costo (g) es constante e igual a uno, indicando que el desplazamiento se realizará sobre un terreno plano y liso.

El segundo factor a tomar en cuenta es la heurística utilizada (h), que difiere de la utilizada en las pruebas 1 y 2. Durante esta prueba se ha considerado la siguiente fórmula como heurística: $h = ((X_2-X_1)^2 + (Y_2-Y_1)^2)^{1/2}$.

La heurística es igual a la *distancia entre dos puntos* $A(X_1,Y_1)$ y $B(X_2,Y_2)$ en el sistema de coordenadas rectangulares. Cada punto corresponde a la esquina superior izquierda de una celda dentro del espacio de configuración. Para verificar la fórmula heurística se puede estudiar la figura 39.

Como AB es la hipotenusa del triángulo rectángulo ABC, el teorema de Pitágoras señala: $AB^2 = AC^2 + CB^2$. Pero: $AC= X_2 – X_1$ y $CB = Y_2 – Y_1$. Por consiguiente: $h = ((X_2-X_1)^2 + (Y_2-Y_1)^2)^{1/2}$.

- **Ejecución**

La disposición de obstáculos es la misma que en la prueba 1, manteniendo los factores constantes, incluyendo el costo de celdas igual a uno. La única variación radica en la heurística utilizada.

El espacio de configuración se puede observar en la siguiente figura:

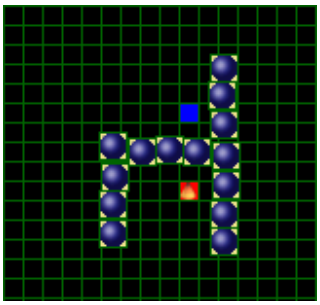


Figura 58. Espacio de configuración prueba 3

La ejecución del algoritmo A* generó los siguientes resultados:

- Posición Inicio en celda 1689
- Posición destino en celda 1409
- Número de celdas sucesoras generadas: 187
- Número de celdas actuales generadas: 142
- Número de celdas que forman parte del resultado: 25 (tomando en cuenta posición de inicio y destino)

La ruta solución generada por el algoritmo se muestra en la siguiente figura:

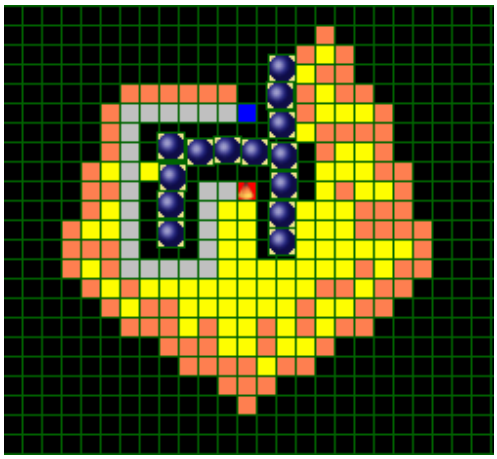


Figura 59. Ruta solución prueba 3

Los resultados para las primeras celdas sucesoras se muestran a continuación:

Celda	X	Y	Enlace	F	G	H
1648	28	27	1649	51.23106	10	41.23106
1709	29	28	1649	60	10	50
1647	27	27	1648	64.72136	20	44.72136
1708	28	28	1648	70.9902	20	50.9902
1769	29	29	1709	80	20	60
1707	27	28	1647	83.85165	30	53.85165
1768	28	29	1708	90.82763	30	60.82763
1829	29	30	1769	100	30	70
1767	27	29	1707	103.2456	40	63.24555
1828	28	30	1768	110.7107	40	70.71068
1889	29	31	1829	120	40	80
1827	27	30	1767	122.8011	50	72.8011
1888	28	31	1828	130.6226	50	80.62258
1890	30	31	1889	130.6226	50	80.62258
1949	29	32	1889	140	50	90

Tabla 5. Extracto celdas sucesoras de prueba 3

Es posible observar de los resultados anteriores como varían los valores de *H* con respecto a los valores de la prueba uno. Se debe recordar que los valores de *H* corresponden a la distancia entre la celda sucesora y la celda marcada como posición destino.

- **Observaciones**

La principal observación que se puede realizar consiste en que el algoritmo A* en estas condiciones, encuentra la ruta más corta. Esta ruta puede compararse con la prueba uno realizada anteriormente. En las dos pruebas los costos se mantienen constantes para todas las celdas del entorno, la disposición y número de obstáculos; sin embargo, en la primera prueba, no encuentra la ruta más corta. La diferencia radica en la elección de la heurística.

Otra observación que puede realizarse, es que el número de celdas sucesoras y actuales es mayor que las obtenidas en la primera prueba.

La complejidad del algoritmo A* radica en la buena elección de una heurística, y en las pruebas realizadas, observamos, que si los costos se mantienen constantes, la heurística utilizada en esta prueba logra encontrar la ruta más corta.

PRUEBA 4

- **Especificaciones**

En la prueba anterior se ha elegido la distancia entre una celda sucesora y la celda destino como heurística, cuyo resultado es mejor que el obtenido utilizando la heurística de la primera prueba. Es en este momento, en donde es posible analizar el impacto de los costos en la función *F* cuando se ha elegido una nueva heurística.

Para ello se ha elegido la siguiente disposición de obstáculos:

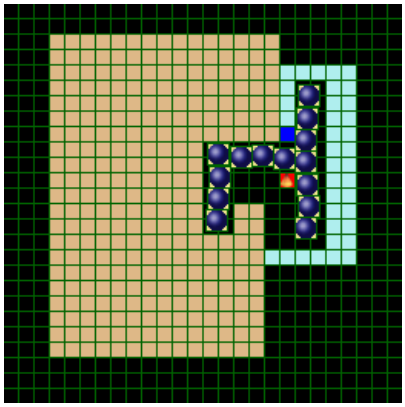


Figura 60. Espacio de configuración con costos prueba 4

En esta prueba se observará como afectan los costos a la función evaluadora del algoritmo. En la prueba anterior se obtuvo que la ruta más corta se encuentra del lado izquierdo de la celda de inicio. Es por ello, que se ha colocado un costo de 10 a lo largo de toda esta trayectoria, que es la más corta. Del lado derecho, correspondiente a la ruta más larga, se ha asignado un costo de 3 a las celdas.

- **Ejecución**

La ejecución del algoritmo A* generó los siguientes resultados:

- Posición Inicio en celda 1590
- Posición destino en celda 1410
- Número de celdas sucesoras generadas: 561
- Número de celdas actuales generadas: 478
- Número de celdas que forman parte del resultado: 36

La solución se muestra a continuación:

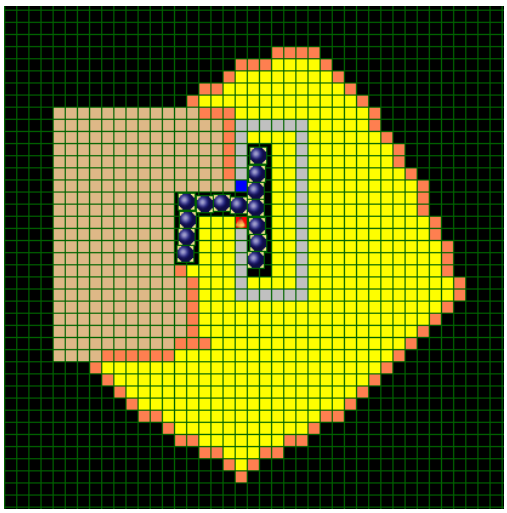


Figura 61. Ruta solución de prueba 4

- **Observaciones**

Como es de notar, la función evaluadora considera tanto la distancia como los costos asociados a las celdas. De esta forma, la ruta solución no es la más corta, pero representa la de menor costo para el desplazamiento del móvil.

La ruta solución es muy parecida a la obtenida en la primera prueba, en la cual la heurística variaba y los costos se mantenían constantes.

CONCLUSIONES

1. La calidad óptima del algoritmo A* depende de la elección de la heurística, es decir, dependiendo de este factor, así se expandirá el menor número de nodos. En la prueba 1 y 3 bajo las mismas condiciones de costos, el menor número de nodos fue expandido en la prueba 1, sin embargo, la heurística no garantizó que se obtuviera la ruta más corta. La calidad óptima del algoritmo A* radica en expandir el menor número de nodos, concepto diferente a encontrar la ruta más corta.

2. Otra característica del algoritmo a ser evaluada es su complejidad, esta se encuentra también ligada a la elección de la heurística. El empleo de una buena heurística se traduce en ahorro en memoria en comparación a una búsqueda no respaldada con información.

3. El algoritmo A* es la combinación de dos funciones de evaluación: g(h) y h(n), encargándose de encontrar la mejor ruta. Esto puede ser visible en la cuarta prueba, en la que es evidente como el algoritmo compara la distancia entre las celdas sucesoras y la celda destino, y el costo de desplazamiento por un terreno. Si bien no encuentra la ruta más corta en esta prueba, si encuentra la de menor costo para la ruta solución.

Anexo No.4 Graficación en Tercera Dimensión utilizando OpenGL

Open Graphics Library (OpenGL) definido como una "interfaz de software para hardware de gráficos", consiste de una librería escrita originalmente en lenguaje C, para el modelado y gráficos en tercera dimensión. Esta librería fue creada considerando la posibilidad de extenderla a cualquier tipo de plataforma y asegurar así su portabilidad y extensibilidad de uso.

Una aplicación basada en OpenGL esta escrita en algún lenguaje de programación que incorpora una o más librerías de OpenGL. Las librerías en común utilizadas por la mayoría de los lenguajes de programación son: gl.h, glu.h y glut.h, en ellas se definen el conjunto de funciones que permiten la graficación en tercera dimensión.

OpenGL no incluye ninguna función para la gestión de ventanas o interacción con el usuario, esto significa que cada entorno en particular (Windows, UNIX, etc.) tiene sus propias funciones para este propósito, y es responsable de facilitar a OpenGL la facultad de dibujar en una ventana.

- **Sistemas gráficos (*dispositivos y elementos*)**

Un sistema gráfico típico se compone de los siguientes elementos físicos:

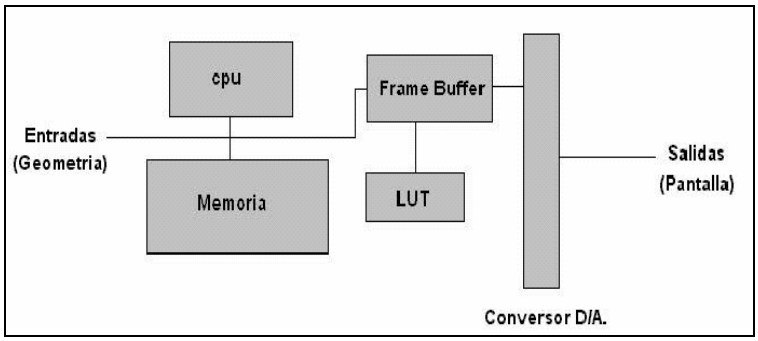


Figura 62. Sistema Gráfico

Las características generales de cada uno de los elementos de la figura anterior son:

- *Entradas*: todo aquello que nuestro programa ha calculado y desea dibujar.

- *Procesador ("CPU")*: máximo administrador del sistema, se encarga de gestionar la comunicación entre todos los módulos.

- *Memoria*: provee el almacenamiento temporal de los cálculos del programa.

- *Frame Buffer:* zona de memoria destinada a almacenar todo aquello que debe ser dibujado. Antes de presentar la información por pantalla, esta se recibe en el frame buffer. Por lo tanto una aplicación basada en OpenGL escribe en esta área de memoria y automáticamente envía su contenido a la pantalla.

- *Look Up Table ("LUT")*: Esta "tabla" contiene todos los colores que tenemos disponibles en nuestro sistema, comúnmente conocida como "paleta". Normalmente cada color tiene un identificador en la tabla y puede referirse a el para ser utilizado en la aplicación grafica. Por ejemplo si se dibuja un polígono de color rojo, se modifica su atributo de color dándole a este el valor del identificador que tiene el color rojo en la LUT.

- *Conversor D/A*: la información contenida en el frame buffer a nivel de bit es digital y por tanto debe convertirse a su homónimo analógico para poder ser procesada por un CRT (tubo de rayos catódicos) y proyectada en la pantalla (monitor).

- Salidas: la información que observamos en nuestro monitor.

Graficación Tridimensional

La representación de objetos en tres dimensiones se lleva a la práctica mediante la visualización de imágenes tridimensionales en la computadora. La graficación tridimensional tiene muchos usos en aplicaciones modernas tales como juegos y simulaciones para el análisis de datos científicos, médicos o usos comerciales.

El término *tres dimensiones* (3D) significa que un objeto es descrito mediante 3 dimensiones de medida: ancho, alto y profundidad, por ejemplo un cilindro. Un objeto en dos dimensiones se representa únicamente por su ancho y su alto, y un objeto en tres dimensiones visto desde la pantalla plana de la computadora, son actualmente imágenes en dos dimensiones que provocan una ilusión de profundidad o tercera dimensión.

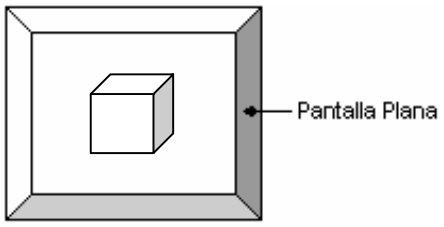


Figura 63. Cubo en 3D visto desde una pantalla plana

La figura anterior muestra un simple cubo en tres dimensiones dibujado con 9 líneas visto desde una pantalla plana. Sin embargo lo que permite mirar tres dimensiones es la perspectiva o el ángulo entre las líneas del cubo que provocan una ilusión de profundidad. Cabe señalar que los ojos del ser humano ven imágenes que el cerebro combina para dar una percepción tridimensional.

Sistema de coordenadas de ventana

El sistema de coordenadas mas común es el sistema de coordenadas cartesianas, es decir de dos dimensiones. Este tipo de coordenadas se expresan mediante una coordenada en X y una coordenada en Y, ambas coordenadas definen el plano XY.

Antes de dibujar objetos en OpenGL, se debe especificar la región del plano cartesiano que ocupa la ventana, dicha región se define como área de recorte (*clipping*). En un espacio en dos dimensiones, el área de recorte es el mínimo y máximo valores de x e y que forman parte de la ventana.

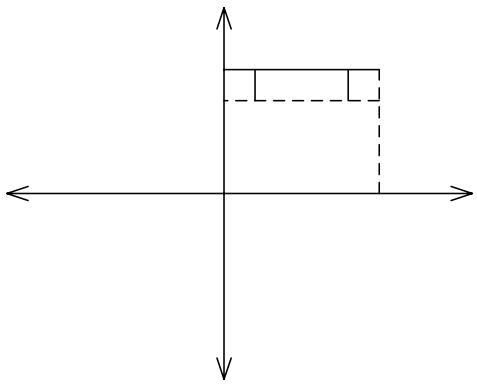


Figura 64. Coordenadas de recorte (clipping)

En la figura anterior puede observarse el rango de la coordenada en x de la ventana (de 0 a 150) y el rango de la coordenada en Y (de 0 a 150).

Muchas veces el área de recorte (ancho y alto) no coincide con el ancho y alto de la ventana en píxeles. Por ello el sistema de coordenadas debe ser convertido a las coordenadas en píxeles de la pantalla. Esta conversión se define como el viewport. El viewport o coordenadas de vista es la región dentro de la ventana del usuario que es utilizada para dibujar el área de recorte, es decir, mapea o convierte el área de recorte a una región de la ventana.

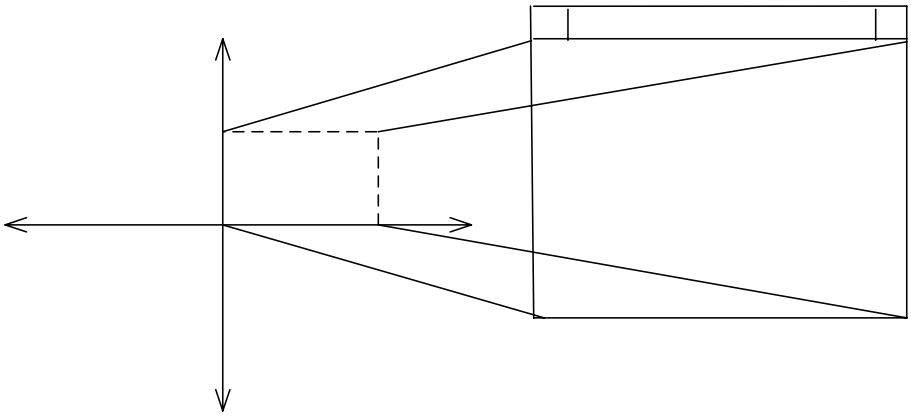


Figura 65. Un viewport definido como dos veces el tamaño del área de recorte

En la figura anterior se muestra una ventana de 300x200 pixeles con coordenadas de vista (viewport) exactamente igual al área de la ventana del usuario. Las coordenadas de vista son utilizadas para comprimir o alargar una imagen dentro de la ventana, y también para mostrar solo una porción del área de recorte en todo el área de la ventana.

Transformaciones de Coordenadas

Las transformaciones hacen posible la proyección de coordenadas en 3D a una pantalla plana (2D). Las transformaciones permiten también rotar, mover, y cambiar el tamaño de los objetos de la escena. En lugar de modificar un objeto directamente, una transformación modifica el sistema de coordenadas. Una vez una transformación rota el sistema de coordenadas, los objetos aparecen rotados al momento de dibujarse. Existen tres tipos de transformaciones que ocurren entre el tiempo cuando se especifican los vértices de un objeto y el tiempo en que estos aparecen en la pantalla: Vistas, Modelado y Proyección.

-x

- **Transformaciones de Vista**

Las transformaciones de vista son las primeras en ser aplicadas a la escena. Son utilizadas para determinar el punto o posición desde donde se vera la escena. Por defecto, el punto de observación esta en el origen (0,0,0) mirando sobre el eje z negativo (hacia adentro de la pantalla del monitor). Cuando el punto de observación es ubicado en el origen, los objetos que se dibuje con valores de z positivo, estarán detrás del observador.

Estas transformaciones permiten además ubicar el punto de observación en cualquier posición y dirección. Determinar la transformación de vista es como ubicar y apuntar una cámara en la escena.

- **Transformaciones de Modelado**

Las transformaciones de modelado son utilizadas para manipular el modelo a graficar, y en particular los objetos dentro de el. Esta transformación mueve los objetos de un lugar a otro, los rota y los puede cambiar de tamaño (cambio de escala).

La siguiente figura muestra las tres transformaciones de modelado que pueden ser aplicados a los objetos:

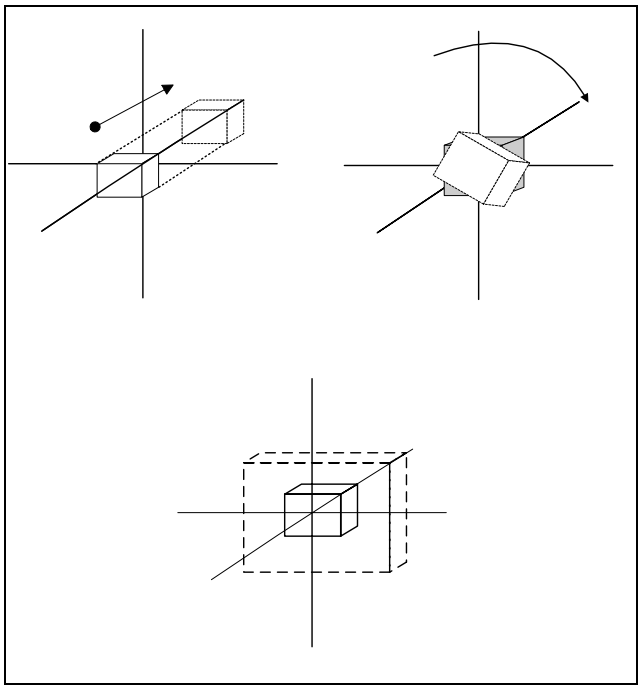


Figura 66. Transformaciones de modelado

Transformaciones de Proyección

Con este tipo de transformación convertimos (proyectamos) las coordenadas 3D de la escena a coordenadas 2D de nuestro plano de proyección (pantalla del monitor).

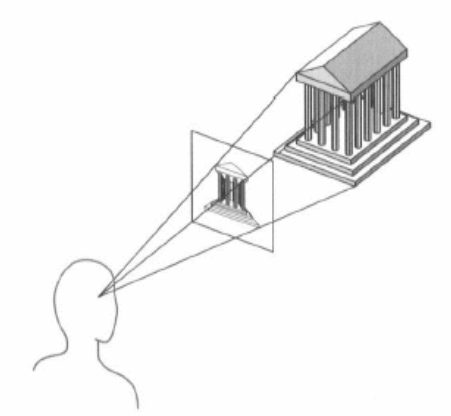


Figura 67 Proyectando una imagen de 3D a 2D

Esta proyección define el volumen de vista y establece planos de recorte, es decir, establece la región del sistema de coordenadas tridimensional que ocupa la ventana donde se observa el modelo.

Los tipos de proyección utilizadas por OpenGL son:

- Proyección Ortográfica: En este tipo de proyección todos los polígonos son dibujados en la pantalla exactamente con las dimensiones relativas especificadas para el modelo. Se utiliza tradicionalmente en proyectos de ingeniería del tipo de programas CAD/CAM.

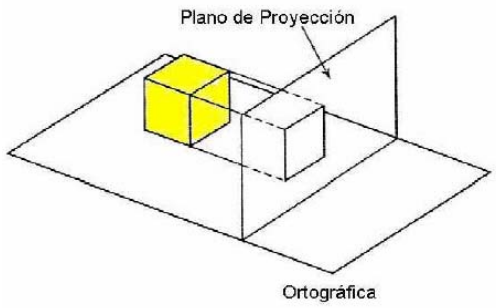


Figura 68 Proyección ortográfica

- Proyección en Perspectiva: En este tipo de proyección los objetos y escenas se muestran tal como son en la vida real. La vista en perspectiva se caracteriza por una disminución del tamaño de la escena, es decir, cuanto más se aleja un objeto del observador, más pequeña se hace su imagen. Esto permite que los objetos más distantes parezcan mas pequeños que los que están mas cerca, aunque sean de la misma medida. Las líneas paralelas no siempre se dibujaran paralelas. En una línea del tren, por ejemplo, los rieles son paralelos, pero con la proyección en perspectiva parecen converger en algún punto en la distancia. A esto se le llama *punto de desvanecimiento*.

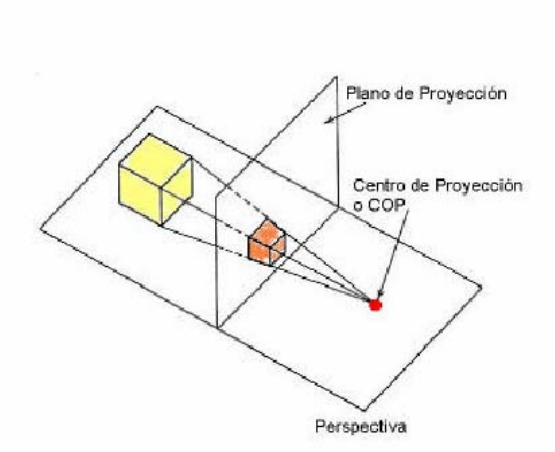


Figura 69 Proyección en perspectiva

- **Efectos aplicados a objetos tridimensionales**

a) Colores

Un color es simplemente una onda de luz que es visible por el ojo humano. Un color en la pantalla de la computadora, se representa por píxeles y es emitido por diferentes intensidades de luz roja, verde y azul. Esta combinación de intensidades es conocida como sistema RGB (por sus siglas en ingles).

En cualquier sistema grafico, cada píxel utiliza la misma cantidad de memoria para almacenar un color. La memoria utilizada por todos los píxeles de una escena es llamada *color buffer*. El tamaño de un buffer es usualmente medido en bits, así que un buffer de 8 bits, podría almacenar 8 bits de datos, es decir, hasta 256 colores diferentes para cada píxel.

OpenGL especifica un color exactamente por la combinación de separadas intensidades de colores rojo, verde y azul. El “cubo de colores” mostrado en la figura 8, contiene todos los posibles colores que pueden ser aplicados ya sea en la superficie del cubo o en su interior.

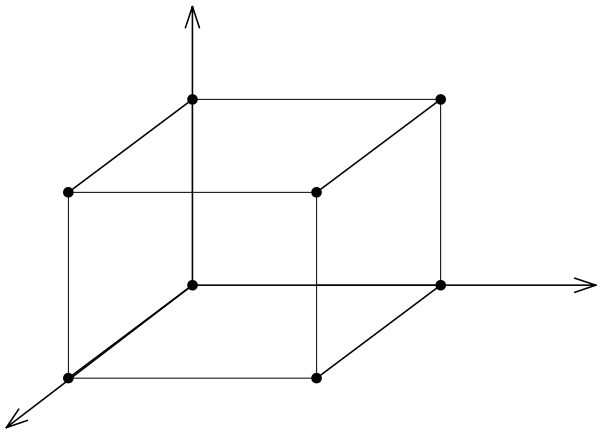


Figura 70 Espacio de colores RGB

b) Luz y Sombra

La iluminación permite que los objetos vistos en tres dimensiones se asemejen más al mundo real. La forma en que los objetos son iluminados depende básicamente de dos características:

- El tipo de luz e intensidad con que se iluminan los objetos en la escena.

- El origen de la luz (o lámpara), es decir, la posición sobre la cual se iluminara la escena.

Cya
(0,255

Los tipos de iluminación soportadas por OpenGL son:

- **Iluminación Ambiental**

Este tipo de luz no proviene de una dirección en particular. Tiene un origen no determinado, pero los rayos de luz tienden a rebotar en la escena convirtiéndose en rayos sin dirección. Objetos iluminados por una luz ambiental llegan a ser uniformemente iluminados en todas sus superficies y direcciones.

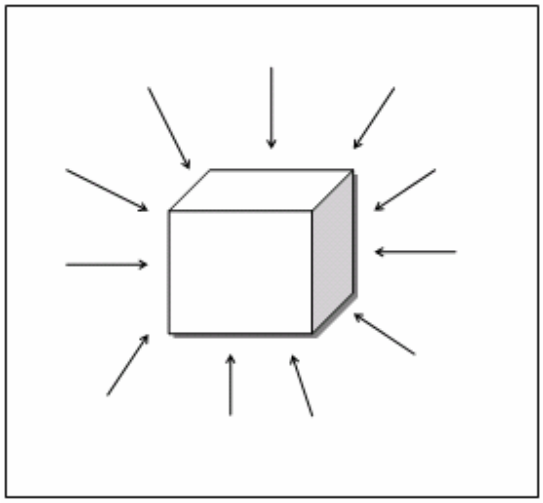


Figura 71 Objeto iluminado por una luz ambiental

- **Efectos de iluminación ambiental**

Anteriormente se menciono que la computadora representa los colores de un píxel en términos de rojo, verde y azul (RGB), así mismo, los efectos de luz ambiental deben expresarse en términos de intensidades RGB. Por ejemplo para un origen de luz con la mitad de intensidad para rojo, verde y azul, se tiene un valor RGB (0.5, 0.5, 0.5) para dicho origen. Si este ambiente de luz ilumina un objeto con propiedades reflectivas especificadas en términos de RGB por ejemplo (0.5, 1.0, 0.5), entonces el color resultante desde el ambiente de luz seria (0.25, 0.5, 0.25), que es el resultado de multiplicar en términos de RGB el origen de luz ambiente por la propiedad ambiental del material.

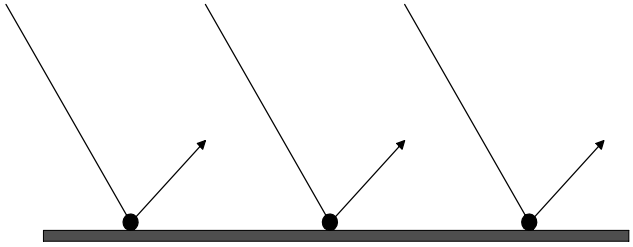


Figura 72 Calculando el color ambiente de un objeto

En conclusión, los componentes de los colores del material son los que determinan el porcentaje de incidencia de luz que se refleja.

Origen de

- **Iluminación Difusa**

Este tipo de luz proviene de una dirección en particular, pero no refleja uniformemente una superficie, es decir, los rayos de luz son dispersos. Un ejemplo de luz difusa seria un foco o proyector y el sol.

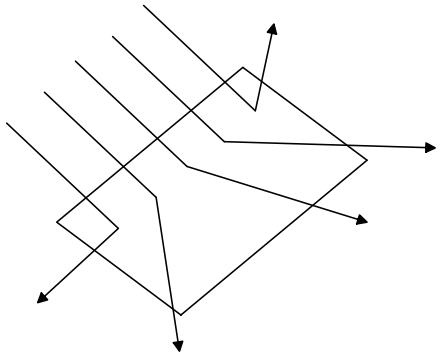


Figura 73 Objeto iluminado por un origen de luz difusa

- **Iluminación Especular**

Se trata de la luz que viene de una dirección particular y rebota sobre un objeto siguiendo una determinada dirección. Es la componente responsable de las zonas más brillantes en la geometría, es decir, tiende a causar manchas brillantes sobre la superficie.

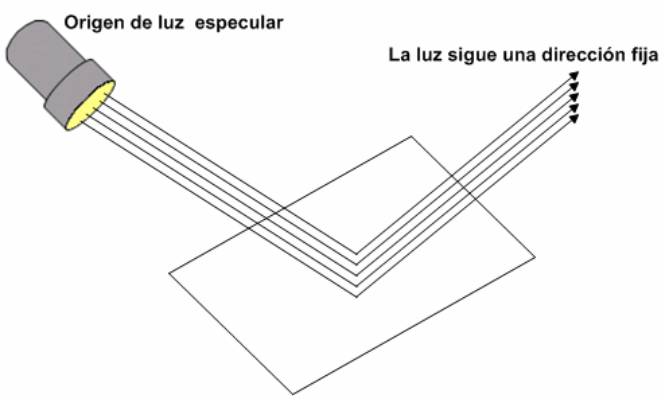


Figura 74 Objeto iluminado por un origen de luz especular

- **Origen de la Luz**

La luz aparte de la intensidad y colores con que refleja la escena, tiene un origen y es determinado por su ubicación y dirección.

Cuando se especifica un origen de luz sobre una superficie o polígono, esta luz crea un ángulo con el plano al rebotar con la superficie. Esta luz es entonces reflejada con cierto ángulo en la dirección de la persona que ve la escena (Ver figura 11). Estos ángulos son usados en conjunto con las propiedades de los materiales e iluminación.

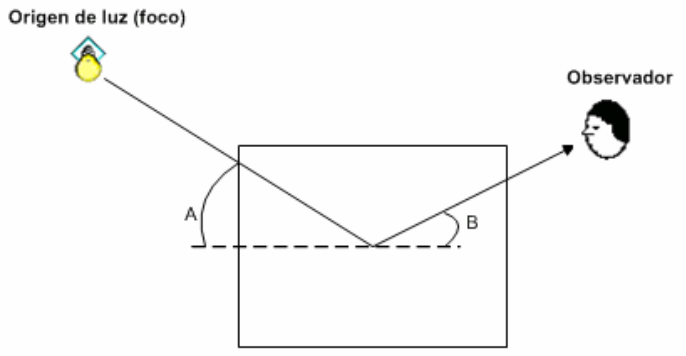


Figura 75 Origen de luz reflejando una superficie con un ángulo específico

Geométricamente no se puede encontrar el ángulo entre un simple punto y una línea en un espacio tridimensional, esto debido a que existe un infinito número de posibilidades. Por esta razón, se debe asociar con cada vértice alguna información clave que denote una dirección hacia arriba desde el vértice y hacia fuera de la superficie. Dicha información se calcula obteniendo el vector normal a un plano.

- **Especificando una Normal**

Para iluminar una superficie o plano, se necesita también información sobre su *vector normal asociado*. La normal de un plano es en realidad un vector perpendicular a este.

Por ejemplo si se quiere calcular el vector normal a la cara superior del objeto de la figura 12 (formado por los vértices A, B, C y D), se haría de la siguiente manera:

Tomando en cuenta que todos estos vértices pertenecen a un mismo plano, entonces para encontrar un vector normal a este plano se debe encontrar un vector que sea perpendicular a cualquiera de estos vértices. Para ello se calculan dos vectores pertenecientes a la cara superior del objeto calculando su *producto*

vectorial. El resultado de esta operación será un vector perpendicular a ambos y por lo tanto una normal del plano.

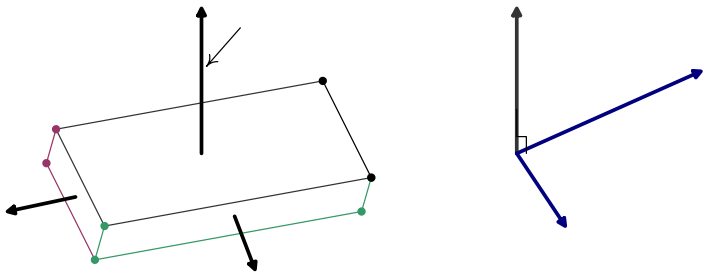


Figura 76 Cálculo de normales para una cara de un objeto

En la figura se puede observar como a partir de tres vértices (A, B y C) Se crean dos vectores que tras su producto vectorial se obtiene el vector normal. Este vector ya puede asociarse a la correspondiente cara.

Mapeo de Texturas

El mapeo de texturas consiste básicamente en rellenar un polígono con imágenes (mapas de bits) en la escena. Las texturas vienen a darle un mayor realismo a las escenas que elaboramos no solo con OpenGL, sino con cualquier motor de gráficos. Estas nos pueden ser útiles para representar de manera sencilla la rugosidad ó la finura de una superficie.

Antes de aplicar una textura a un polígono, debe definirse la imagen que servirá como textura. Estas imágenes siguen las mismas reglas de almacenamiento que los bitmaps.

A

OpenGL soporta dos técnicas de texturizado: Una es por Manipulación del Color de la Superficie y la otra es por Mapeo del Ambiente. La primera de ellas se refiere a manipular directamente los colores de las superficies de las caras a afectar, sin tomar en cuenta nada más que el color de la textura a aplicar. La segunda

consiste en involucrar los colores de los materiales de los objetos a texturizar, así como las luces y las normales obteniendo una textura combinada con el ambiente.

OpenGL soporta imágenes de texturas de una y dos dimensiones (1D y 2D) y cuyas medidas sean *una potencia de dos*. Por ejemplo: imágenes bidimensionales de 64 X 64 píxeles, de 256 X 128, etc. En algunas implementaciones de OpenGL se han extendido y se soportan texturas de 3 y 4 dimensiones también (3D y 4D).

Las transformaciones de modelado, que incluye rotaciones, traslaciones y escala, pueden ser aplicadas a una textura.

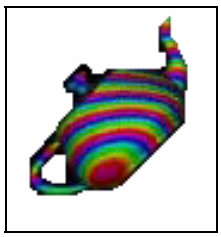


Figura 77 Textura aplicada a un objeto 3D

- **Matrices y Vectores en OPENGL**

Toda la geometría que se despliega en las aplicaciones OpenGL está basada en los conceptos de Matrices y Vectores, y las operaciones aritméticas aplicables a estas estructuras.

En OpenGL existen básicamente tres matrices principales:

- Una matriz de proyección llamada GL_PROJECTION, la cual nos permite determinar la perspectiva que usaremos para observar la escena generada, así como el tipo de proyección a usar (Transformación de proyección).

- Una matriz de Modelado llamada GL_MODELVIEW, la cual sirve para aplicar a nuestra escena operaciones de rotación, traslación o escalamiento, o bien para manipular la posición y orientación de la cámara, obteniendo así las animaciones.

- Una matriz para el manejo de Texturas llamada GL_TEXTURE, sobre la cual también podemos aplicar las transformaciones lineales de rotación, traslación y escalamiento para manipular las texturas a utilizar en las figuras de nuestra escena.

Se puede hacer uso de cada una de estas matrices mediante el procedimiento GLMatrixMode(), el cual permite seleccionar una de estas matrices para su configuración. Para inicializar con valores cada matriz se invoca a un procedimiento llamado glLoadIdentity(), el cual carga la matriz identidad, cuyos valores son 1 en toda su diagonal lo cual hace que multiplicar cualquier vector por esta matriz nos dé como resultado al mismo vector sin que haya sufrido ninguna transformación.

Los vectores son un conjunto de valores que nos permiten definir dentro de nuestra escena, desde los vértices de las figuras, hasta los colores, los materiales y las luces, entre otras muchas cosas.

Existen básicamente dos formas de trabajar con vectores en OpenGL: Una es con sus elementos como variables independientes, y otra manejarlos como una estructura de datos. Cuando los elementos del vector se manipulan de forma independiente, cada vector es descompuesto en 3 ó 4 valores según sea el caso. Cuando los elementos del vector se manipulan como una estructura, los valores están contenidos en una estructura de datos que bien puede ser un arreglo ó un registro.

Los vectores pueden descomponerse en 3 o 4 valores así como ocurre en los colores, que están compuestos por tres valores que determinan el color codificados en RGB, y un elemento extra (en algunos casos) que determina el nivel de transparencia para el objeto, cara ó vértice al cual corresponda dicho color.

Anexo No.5 Librerías CsGL

Las librerías de enlace dinámico CsGL fueron creadas por el "Equipo de desarrollo CsGL" con el fin de desarrollar aplicaciones utilizando OpenGL y el .Net Framework creado por Microsoft. Provee soporte a OpenGL 1.1-1.4 y posee numerosas extensiones.

Características

- Ambiente: Win32 (Microsoft Windows)
- Licencia : licencia BSD (Berkeley Software distribution)
- Lenguaje : Inglés
- Sistemas operativos: Windows 95/98/NT/2000/XP
- Lenguajes de programación: C, C#, C++, Java, Visual Basic

Descripción

Fueron creadas dos librerías de enlace dinámico:

- csgl.dll : es un assembly de .Net que provee clases y utilerías de OpenGL.
- csgl.native.dll : es una librería de enlace dinámico que desarrolla algunos trabajos propios de CsGL para funcionar.

CsGL implementa una "envoltura" para la librería OpenGL permitiendo a cualquier lenguaje .NET utilizar dichas librerías.

Posee las siguientes características:

- OpenGL 1.4 API y extensiones
- Un generador de extensiones en in <csgl>/extras/generator/generator.exe.
- Generador de tipos de letra en CsGL.OpenGL.GDITextureFont.

- Clases para uso del teclado y del ratón (CsGL.Util.Mouse & CsGL.Util.Keyboard)
- Ventana para cambio de resolución de pantalla (CsGL.Util.ScreenForm).

REQUERIMIENTOS

- Librerías CsGL que pueden encontrarse en un archivo comprimido con extensión *.zip* que contiene las diferentes DLL.Inluye también un archivo de instalación de las mismas.
- Las librerías de OpenGL
- .NET framework

REFERENCIA Y SOPORTE

Equipo de desarrollo CsGL
http://csgl.sourceforge.net