**El METODO DE DISEÑO DE LA INGENIERIA**

**Contexto del problema**

El contexto de este problema se enmarca en el ámbito del desarrollo de software y diseño de juegos. El objetivo es crear un juego interactivo que se base en un grafo, una estructura de datos matemática que consta de vértices y aristas. En este caso, se requiere un grafo con un mínimo de 50 vértices y 50 aristas, lo que implica una estructura de juego de cierta complejidad. Este grafo debe ser la columna vertebral del juego y permitir la aplicación de al menos dos algoritmos de grafos, que pueden ser seleccionados entre un conjunto específico, como Recorridos sobre Grafos (BFS, DFS), Caminos de Peso Mínimo (Dijkstra, Floyd-Warshall) y Árbol de Recubrimiento Mínimo (Prim, Kruskal).m Además, se establecen requisitos mínimos estrictos, que incluyen el desarrollo de dos versiones diferentes del grafo, cada una de las cuales debe seguir un proceso completo.

En este contexto, el videojuego en desarrollo es una experiencia tridimensional ambientada en el espacio, donde el jugador asume el papel de un piloto espacial en busca de un planeta distante. Para alcanzar dicho destino, el usuario debe navegar a través de un complejo entramado de caminos y rutas representados por el grafo mencionado anteriormente. El grafo actúa como la red interconectada de sistemas estelares, y cada vértice representa un punto clave en el espacio, como estaciones espaciales o asteroides. Las aristas, por otro lado, simbolizan las rutas navegables entre estos puntos.

El objetivo principal del juego es recolectar naves espaciales dispersas en distintos puntos del grafo para fortalecer la flota del jugador y aumentar sus posibilidades de éxito. Sin embargo, la travesía no será fácil, ya que el espacio está poblado por enemigos hostiles que el jugador debe esquivar o enfrentar estratégicamente. La complejidad del grafo y su aplicación en el juego añaden un nivel adicional de desafío, ya que el jugador debe planificar cuidadosamente sus movimientos para optimizar la recolección de naves y evitar confrontaciones innecesarias

**FASE 1: IDENTIFICACIÓN DEL PROBLEMA**

Problema: El problema se centra en el desarrollo de un juego interactivo bidimensional ambientado en el espacio, donde el jugador debe llegar a un planeta recolectando naves espaciales dispersas.

Necesidades:

* Diseño de Entorno Espacial
* Modelado de Naves Espaciales
* Mecánicas de Recolección
* Objetivo del Juego
* Interfaz de Usuario Intuitiva
* Efectos Visuales y Sonoros
* Adaptabilidad del Nivel de Dificultad
* Pruebas y Retroalimentación
* Narrativa del Juego

**TABLA DE ESPECIFICACIÓN DEL PROBLEMA DE INGENIERÍA DE SOFTWARE**

|  |  |
| --- | --- |
| CLIENTE |  |
| USUARIO |  |
| REQUERIMIENTOS FUNCIONALES | 1. Exploración Espacial 2. Recolección de Naves Espaciales 3. Ubicación inicial 4. Movilidad del avatar 5. Combate Espacial 6. Navegación Estratégica |
| CONTEXTO DEL PROBLEMA |  |
| REQUERIMIENTOS NO FUNCIONALES | 1. Rendimiento 2. Compatibilidad 3. Estética Visual 4. Sonido Ambiente 5. Intuitividad en los Controles: 6. Seguridad 7. Estabilidad y Rendimiento |

| Nombre o identificador | 1. **Exploración Espacial** | | |
| --- | --- | --- | --- |
| Resumen | El juego debe permitir al jugador explorar un entorno espacial tridimensional, interactuando con diversos elementos como planetas, asteroides y estaciones espaciales. | | |
| Entradas | Nombre entrada | Tipo de dato | Condición de selección o repetición |
| controles | Char |  |
| Resultado o postcondición | El jugador se encuentra inmerso en un entorno espacial visualmente detallado y puede explorar libremente, observando planetas, asteroides y estaciones espaciales en tiempo real. | | |

| Nombre o identificador | 1. **Recolección de Naves Espaciales** |
| --- | --- |
| Resumen | El juego debe Proporcionar al jugador la tarea de recolectar naves espaciales dispersas en el entorno para fortalecer su flota |
| Resultado o postcondición | Al recolectarla, la nave se agrega al inventario del jugador, mejorando su flota |

| Nombre o identificador | 1. **Ubicación inicial** |
| --- | --- |
| Resumen | El Juego debe establecer una ubicación inicial para el jugador dentro del entorno espacial al comenzar el juego. |
| Resultado o postcondición | El avatar del jugador está posicionado en la ubicación inicial designada, listo para comenzar la exploración. Se ha establecido el contexto narrativo inicial |

| Nombre o identificador | 1. **Movilidad del avatar** |
| --- | --- |
| Resumen | El juego debe permitir al jugador moverse de manera fluida y controlar su avatar en el espacio tridimensional. |
| Resultado o postcondición | El avatar se mueve suavemente en la dirección especificada por el jugador, con ajustes de velocidad y dirección según las preferencias del jugador. |

| Nombre o identificador | 1. **Combate Espacial** |
| --- | --- |
| Resumen | El juego debe introducir un sistema de combate que permita al jugador enfrentarse a enemigos hostiles durante la exploración espacial. |
| Resultado o postcondición | Después de realizar acciones de combate, el estado visual y auditivo del juego refleja el resultado del enfrentamiento. Puede incluir la destrucción de enemigos o cambios en la salud de las naves |

| Nombre o identificador | 1. **Navegación Estratégica** |
| --- | --- |
| Resumen | El juego debe facilitar la capacidad del jugador para planificar rutas estratégicas a través del grafo, teniendo en cuenta la ubicación de naves espaciales, enemigos y obstáculos. |
| Resultado o postcondición | El jugador ha seleccionado una ruta estratégica a través del grafo, y el juego proporciona visualizaciones claras de la ruta y posiblemente ajusta la posición del jugador en consecuencia. |

**FASE 2: RECOPILACIÓN DE LA INFORMACIÓN NECESARIA**

**Adjacency matrix** La matriz de adyacencia es una matriz cuadrada que se utiliza como una forma de representar relaciones binarias.

1. Por cada arista que une a dos nodos, se suma [1](https://es.wikipedia.org/wiki/Uno) al valor que hay actualmente en la ubicación correspondiente de la matriz.
   * Si tal arista es un [bucle](https://es.wikipedia.org/wiki/Bucle_(teor%C3%ADa_de_grafos)) y el grafo es [no dirigido](https://es.wikipedia.org/wiki/Grafo#Grafo_no_dirigido), entonces se suma 1 o [2](https://es.wikipedia.org/wiki/Dos) (dependiendo de la convención usada).
   * Si el grafo es [ponderado](https://es.wikipedia.org/wiki/Grafo_ponderado), entonces en lugar de un 1 se suma el peso de la arista respectiva.

Finalmente, se obtiene una matriz que representa el número de aristas (relaciones) entre cada par de nodos (elementos).

Existe una matriz de adyacencia única para cada grafo (sin considerar las [permutaciones](https://es.wikipedia.org/wiki/Permutaci%C3%B3n) de filas o columnas), y viceversa.

* Para un grafo no dirigido la matriz de adyacencia es simétrica.
* El número de caminos *Ci,j*(*k*), atravesando *k* aristas desde el nodo *i* al nodo *j*, viene dado por un elemento de la potencia k-ésima de la matriz de adyacencia:

**Grafo** Un grafo ​ es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto

**Adjacendy list** Una lista de adyacencia es una representación de todas las [aristas](https://es.wikipedia.org/wiki/Arista_(teor%C3%ADa_de_grafos)) o arcos de un [grafo](https://es.wikipedia.org/wiki/Grafo) mediante una [lista](https://es.wikipedia.org/wiki/Lista_(inform%C3%A1tica)). Si el grafo es no dirigido, cada entrada es un [conjunto](https://es.wikipedia.org/wiki/Conjunto) o [multiconjunto](https://es.wikipedia.org/wiki/Multiconjunto) de dos [vértices](https://es.wikipedia.org/wiki/V%C3%A9rtice_(teor%C3%ADa_de_grafos)) conteniendo los dos extremos de la arista correspondiente. Si el grafo es dirigido, cada entrada es una [tupla](https://es.wikipedia.org/wiki/Tupla) de dos nodos, uno denotando el nodo fuente y el otro denotando el nodo destino del arco correspondiente. Típicamente, las listas de adyacentes no son ordenadas

**BFS**  Es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo, comenzando en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo), para luego explorar todos los vecinos de este nodo. Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo. A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

**DFS** Es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) de [búsqueda no informada](https://es.wikipedia.org/wiki/B%C3%BAsquedas_no_informadas) utilizado para recorrer todos los nodos de un [grafo](https://es.wikipedia.org/wiki/Grafo) o [árbol (teoría de grafos)](https://es.wikipedia.org/wiki/%C3%81rbol_(teor%C3%ADa_de_grafos)) de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa ([Backtracking](https://es.wikipedia.org/wiki/Backtracking" \o "Backtracking)), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

**Algoritmo Dijkstra**  Es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) para la determinación del [camino más corto](https://es.wikipedia.org/wiki/Problema_del_camino_m%C3%A1s_corto), dado un [vértice](https://es.wikipedia.org/wiki/V%C3%A9rtice_(teor%C3%ADa_de_grafos)) origen, hacia el resto de los vértices en un [grafo](https://es.wikipedia.org/wiki/Grafo) que tiene pesos en cada [arista](https://es.wikipedia.org/wiki/Arista_(teor%C3%ADa_de_grafos)). La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene. Se trata de una especialización de la [búsqueda de costo uniforme](https://es.wikipedia.org/wiki/B%C3%BAsqueda_de_costo_uniforme) y, como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

**Algoritmo Floyd-Warshall**

1. **El algoritmo de Warshall:** El algoritmo de Warshall es un ejemplo de algoritmo booleano. A partir de una tabla inicial compuesta de 0`s (no hay correspondencia inicial en el grafo) y 1`s (hay una correspondencia, llamase “flecha”, entre nodos), obtiene una nueva matriz denominada “Matriz de Clausura Transitiva” en la que se muestran todas las posibles uniones entre nodos, directa o indirectamente. Es decir, si de “A” a “B” no hay una “flecha”, es posible que si haya de “A” a “C” y luego de “C” a “B”. Luego, este resultado se verá volcado en la matriz final.
2. **El algoritmo de Floyd:** El algoritmo de Floyd es muy similar, pero trabaja con grafos ponderados. Es decir, el valor de la “flecha” que representamos en la matriz puede ser cualquier número real o infinito. Infinito marca que no existe unión entre los nodos. Esta vez, el resultado será una matriz donde estarán representadas las distancias mínimas entre nodos, seleccionando los caminos más convenientes según su ponderación (“peso”).

El algoritmo de **Floyd-Warshall** compara todos los posibles caminos a través del grafo entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo V³ comparaciones (esto es notable considerando que puede haber hasta V2 aristas en el grafo, y que cada combinación de aristas se prueba). Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima.

**Algoritmo Prim** El algoritmo de Prim permite encontrar un árbol recubridor mínimo de un grafo. En otras palabras, el algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices, donde el p eso total de todas las aristas en el árbol es el mínimo posible.

**Algoritmo de Kruskal** Es un proceso que permite unir todos los nodos de un grafo formando un árbol, tomando en cuenta el peso de las aristas y cuyo coste total es el mínimo posible