Método de la ingeniería

FASE 1:

Contexto del problema

La empresa "Juegos Épicos" está buscando contratar a un par de estudiantes de informática para diseñar y desarrollar su juego titulado "Space Adventure". Este juego es un juego de exploración planetaria y gestión de recursos en el que el personaje principal, Ramiel, un explorador espacial, está en busca de su gato llamado Naranjita. Ramiel debe viajar a través de muchos planetas en su nave espacial mientras administra sus recursos, como el combustible y el oxígeno, para asegurarse de encontrar a su gato. Para desarrollar el juego de la mejor manera, la empresa busca que los desarrolladores implementen una estructura de datos llamada grafos.

Más información: Space Adventure

Definición del problema

El problema se centra en el desarrollo de un juego llamado "Space Adventure", donde el objetivo principal es ayudar a Ramiel a encontrar a su gato, Naranjita, en un universo compuesto por 50 o más planetas. El objetivo es diseñar e implementar una estructura de datos basada en grafos para gestionar la conectividad entre planetas, así como algoritmos de búsqueda y árbol de recubrimiento mímino para calcular rutas eficientes para Ramiel. Además, es importante tener en cuenta las reglas del juego, la gestión de recursos y la interacción del personaje con el entorno.

A continuación

Cliente	Juegos Épicos
Usuario	Usuario
Contexto del Problema	*Juegos Épicos desea que su juego Space Adventure se implemente con la estructura de datos grafos. Los nodos del grafo serán las estaciones y los planetas del universo, las aristas representarán los caminos entre estaciones y planetas. El juego debe permitirle al jugador viajar entre planetas, para esto se utilizaran algoritmos de búsqueda en grafos. Debe permitirle administrar sus recursos y representar sus estados de salud y oxígeno. Con un árbol de recubrimiento mínimo, le debe permitir al usuario utilizar el modo automático de navegación entre planetas.*
Requerimientos Funcionales	El sistema debe permitirle al usuario: **RF1.** Iniciar juego. **RF2.** Navegar entre planetas o estaciones. **RF3.** Comprar un objeto. **RF4.** Activar piloto automático. **RF5.** Manual de instrucciones

	RF6. Salir.
Requerimientos No Funcionales	- La interfaz de usuario debe ser intuitiva y fácil de usar. - El diseño de los personajes y objetos deben tener un patrón de color y formas que sean “agradables” de ver.

Identificador y nombre	*RF1. Iniciar juego*		
Resumen	*El sistema deberá poder iniciar un juego nuevo para el jugador cuando él lo desee. El juego nuevo debe crear 40 planetas y 10 estaciones de repostaje además, debe interconectarlos con unos caminos y asignarle a esos caminos un peso. Dentro de cada planeta, se debe generar un objeto de recolección o un enemigo. Por otro lado, al jugador se le debe poner sus atributos de vida y a la nave se genera con su combustible al completo. Además, el sistema deberá generar los objetos que venderán las estaciones junto a sus precios. Por último, se deberá poner en una ubicación random (un planeta) al jugador y al gato. Cada uno debe estar en posiciones diferentes*		
Entradas	**Nombre de entrada**	**Tipo de dato**	**Condición de valor válido**
Resultado o postcondición	El sistema generó los planetas y estaciones, además, agregó conexiones entre los planetas y estaciones junto a sus pesos de viaje. Por otro lado, se generaron los objetos dentro de los planetas y los objetos dentro de las estaciones. Además, se generó al jugador junto a sus atributos y su cohete con sus atributos en un nodo random. Por último, el gato también se generó en un planeta random.		
Salidas	**Nombre de la salida**	**Tipo de dato**	**Formato**
	alert	String	*El sistema muestra al jugador el universo explorable.*

Identificador y nombre	*RF2. Navegar entre planetas o estaciones*		
Resumen	*El sistema debe permitirle al jugador poder navegar entre planetas o estaciones. El jugador deberá pulsar con el mouse el nodo que desea visitar, cuando lo pulsa, se obtiene el nombre del planeta o estación y se calcula el costo de viaje y se le informa al usuario para que el decida si desea viajar o no o, se le informa si el planeta elegido está muy lejos del actual.*		
Entradas	**Nombre de entrada**	**Tipo de dato**	**Condición de valor válido**
	planetName	String	*Ingresa el nombre del planeta, se permitirán caracteres y números.*

Resultado o postcondición	El sistema encuentra o no el planeta, si no lo encuentra, le informa al usuario que el planeta no está suficientemente cerca para viajar, si lo encuentra, transporta al personaje al mundo o estación y le resta al combustible del cohete la cantidad de combustible gastado.		
Salidas	**Nombre de la salida**	**Tipo de dato**	**Formato**
	alert	String	*El sistema informa si el jugador pudo viajar o no a el nuevo nodo.*

Identificador y nombre	*RF3. Comprar un objeto*		
Resumen	*Cuando el jugador se encuentra en un nodo que es de tipo estación y lo decide visitar, el sistema debe desplegar una ventana con una interfaz de tienda, donde aparecerán 3 tipos de objetos junto a sus precios. El jugador solo podrá comprar 1 corazón por 2 monedas y luego aparecerá como agotado, el jugador podrá comprar paquetes de 3 balas por 1 moneda, podrá comprar hasta 3 paquetes y por último, el jugador podrá comprar 5 de combustible por 1 moneda y no podrá comprar más de 50 de gasolina. El jugador le da click en lo que quiere comprar junto a su cantidad.*		
Entradas	**Nombre de entrada**	**Tipo de dato**	**Condición de valor válido**
	objectAmount	int	*Solo se permiten números enteros positivos.*
	objectBought	ObjectType	*Solo se permiten los valores: "GUN" o "HEART" o "GASOLINE"*
Resultado o postcondición	El sistema hizo los cálculos de cantidad y tipo de objeto y se lo resto a las monedas del jugador, si las monedas no le alcanzan, el sistema debe avisar al jugador, si no, cambia el interfaz del usuario según sea la compra, sumandole la gasolina al cohete, sumándole una vida o sumándole la cantidad de balas compradas.		
Salidas	**Nombre de la salida**	**Tipo de dato**	**Formato**
	alert	String	El sistema muestra la nueva interfaz del jugador con los cambios realizados según la compra.
	noCoins	String	El sistema despliega un aviso que le avise al usuario que no tiene suficientes monedas

Identificador y nombre	*RF4. Activar piloto automático*

Resumen	*Cuando el jugador se encuentra en cualquier nodo, podrá activar el modo piloto automático con un botón , cuando lo activa el sistema deberá preguntarle cuánta gasolina desea que su cohete gaste en el viaje. Si la cantidad puesta por el jugador es mayor a la que tiene el cohete, se le deberá decir al jugador que no tiene gasolina suficiente, si la cantidad puesta por el jugador deja al jugador con menos de 10 de combustible, se le debe avisar al jugador que estos son sus últimos recursos. Si el jugador decide viajar, se le restará la gasolina según sea el viaje alcanzado por el cohete.*		
Entradas	**Nombre de entrada**	**Tipo de dato**	**Condición de valor válido**
	planetName	String	*Ingresa el nombre del planeta, se permitirán caracteres y números.*
	gasolineAmount	int	*Solo se permiten números enteros positivos .*
Resultado o postcondición	El sistema le informó al usuario si se pudo realizar o no el viaje automático, si lo hizo, se mostró el viaje del cohete por la interfaz de jugador si no, se le mostró un aviso de que no se logró viajar. Si el viaje se realizó pero sobró combustible se lo informa al usuario y se suma en la cantidad de combustible del cohete.		
Salidas	**Nombre de la salida**	**Tipo de dato**	**Formato**
	alert	String	El sistema muestra la nueva interfaz del jugador con los cambios realizados en el movimiento del cohete.
	tripNoCompleted	String	El sistema despliega un aviso que avisa al usuario que no se pudo realizar el viaje.

Identificador y nombre	*RF5. Manual de instrucciones*		
Resumen	*Cuando el jugador se encuentra en el menú principal del juego, el podrá elegir la opción de instrucciones del juego, se le mostrará una ventana con todos los detalles del juego. La ventana tendrá un scroll para ir leyendo el texto y tendrá un botón para ir al menú principal.*		
Entradas	**Nombre de entrada**	**Tipo de dato**	**Condición de valor válido**
Resultado o postcondición	El sistema mostró la ventana con las instrucciones del juego y le mostró al jugador un botón para ir al menú principal.		
Salidas	**Nombre de la salida**	**Tipo de dato**	**Formato**

	alert	String	El sistema muestra una ventana al jugador con instrucciones del juego.

Identificador y nombre	*RF6. Salir*		
Resumen	*Cuando el jugador se encuentra en el menú principal del juego, el jugador podría elegir la opción salir que lo saque del juego y termine la ejecución del juego.*		
Entradas	**Nombre de entrada**	**Tipo de dato**	**Condición de valor válido**
Resultado o postcondición	El sistema sacó al jugador de la ventana de menú principal y terminó la ejecución.		
Salidas	**Nombre de la salida**	**Tipo de dato**	**Formato**

FASE 2:

Con el objetivo de obtener una comprensión completa de los conceptos involucrados, se realiza una búsqueda de definiciones de términos tanto teóricos como prácticos.

- **Teoría de grafos:**

  La teoría de grafos, una disciplina matemática e informática, se dedica al estudio de estructuras llamadas "grafos". Un grafo consta de dos componentes principales: vértices (o nodos), que son los puntos en el grafo, y aristas (o enlaces), que representan las conexiones entre los vértices. Estas conexiones pueden tener direcciones específicas o ser no dirigidas. Los fundamentos de la teoría de grafos se encuentran en la matemática discreta y aplicada, y abarca conceptos de diversas áreas como combinatoria, álgebra, probabilidad, geometría de polígonos, aritmética y topología.

  Aunque la teoría de grafos tiene sus raíces en el siglo XVIII con el famoso problema de los puentes de Königsberg, su influencia y aplicaciones han crecido de manera significativa en la era moderna, especialmente en campos como la informática, las ciencias de la computación y las telecomunicaciones.

  Leonhard Euler, en 1736, fue un pionero al abordar el problema de los puentes de Königsberg, proporcionando el primer resultado significativo en la teoría de grafos. Su trabajo marcó un hito al ofrecer una solución al problema y se considera uno de los primeros logros topológicos en geometría, destacando por no depender de mediciones específicas.

  Gustav Kirchhoff llevó la teoría de grafos más allá al aplicarla al análisis de redes eléctricas. Publicando sus leyes de los circuitos en el contexto de la teoría de grafos, conocidas como las leyes de Kirchhoff, logró una de las primeras aplicaciones prácticas

de esta teoría en ingeniería. Estas leyes proporcionan métodos para calcular voltajes y corrientes en circuitos eléctricos, marcando un hito en la integración exitosa de la teoría de grafos en problemas del mundo real.

- **Grafo ponderado:**

  Un grafo ponderado se caracteriza por asignar valores o pesos a sus aristas. Este tipo de grafos resulta especialmente útil en la representación de redes sociales o estructuras complejas que involucran relaciones valoradas. En el ámbito del análisis de redes sociales y la ciencia de redes, estos grafos se emplean para medir y registrar la importancia de distintos enlaces, proporcionando así una representación más detallada que la ofrecida por los grafos sin pesos.

  Cada arista en un grafo ponderado lleva consigo un peso, permitiendo establecer relaciones entre diversas métricas de la red, también conocidas como conceptos de red, estadísticas o índices. De este modo, las medidas clásicas de centralidad, como el grado, la cercanía o la intermediación, pueden ser adaptadas para considerar los pesos asociados a las aristas. Además, tanto el coeficiente de agrupamiento global como el local pueden aplicarse teniendo en cuenta ternas de valores o fórmulas algebraicas.

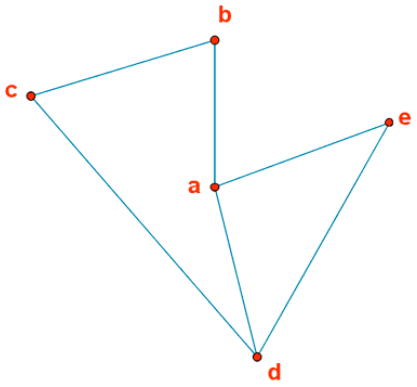
  En el contexto de los grafos ponderados, las trayectorias basadas en la noción de camino pueden definirse, incorporando conceptos bien definidos de valor y longitud de un camino, así como de accesibilidad entre vértices. En resumen, la capacidad de capturar información adicional hace que los grafos ponderados sean herramientas valiosas en diversas aplicaciones, destacando en entornos como las redes sociales y las estructuras complejas.

- **Representación de grafos**

  - **Matriz de adyacencia**

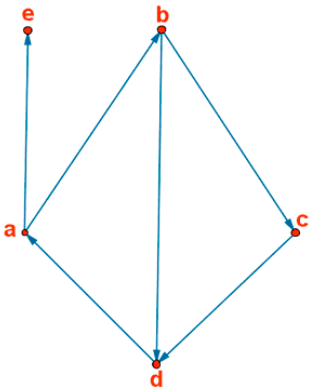
    En la representación de un grafo mediante una matriz de adyacencia, se utiliza una matriz cuadrada para indicar las conexiones entre los nodos del grafo. Cada fila y columna de la matriz representa un nodo, y el valor en la posición (i, j) indica si hay una arista entre el nodo i y el nodo j. Si el grafo es no dirigido y no ponderado, los valores de la matriz son típicamente booleanos, donde "1" indica la presencia de una arista y "0" indica la ausencia de una arista.La complejidad espacial de esta representación es O(V^2), lo que significa que se requiere una gran cantidad de memoria para grafos grandes.

    Ejemplo: Matriz de adyacencia de un grafo no dirigido



	a	b	c	d	e
a	0	1	0	1	1
b	1	0	1	0	0
c	0	1	0	1	0
d	1	0	1	0	1
e	1	0	0	1	0

Ejemplo: Matriz de adyacencia de un grafo dirigido

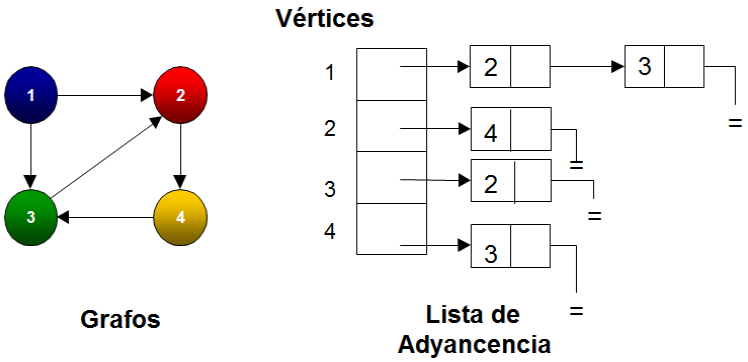


	a	b	c	d	e
a	0	1	0	0	1
b	0	0	1	1	0
c	0	0	0	1	0
d	1	0	0	0	0
e	0	0	0	0	0

- **Lista de adyacencia**

  La lista de adyacencia consiste en una lista de tamaño V, donde cada elemento de la lista representa un vértice y contiene una sublista con los vértices adyacentes a dicho vértice. Esta forma de representación resulta beneficiosa en el contexto de grafos dispersos, caracterizados por la ausencia de conexiones entre la mayoría de los vértices. La complejidad espacial asociada a esta representación es de orden O(V+E), siendo E el número de aristas presentes en el grafo.

  Ejemplo:



- **Recorridos sobre grafo**

- **BFS**

  El algoritmo de búsqueda en amplitud, comúnmente conocido como BFS, es una técnica utilizada para explorar y recorrer grafos y árboles. Este método sigue una estrategia sistemática y nivelada para visitar nodos en el grafo.

  Comienza seleccionando un nodo de inicio y utiliza una cola para organizar los nodos a explorar. El proceso consiste en sacar un nodo de la cola, explorar sus nodos vecinos y agregar aquellos que aún no han sido visitados a la cola. Este proceso se repite hasta que la cola esté vacía, asegurando que se exploren todos los nodos a una distancia determinada antes de pasar a los nodos en la siguiente distancia.

  BFS es valioso en diversas aplicaciones, como la búsqueda de caminos más cortos, la detección de ciclos y la evaluación de la conectividad en grafos. Su enfoque nivelado garantiza que todos los nodos cercanos se visiten antes de extender la exploración a nodos más alejados. Este algoritmo se utiliza comúnmente cuando se busca encontrar la ruta más corta entre dos nodos en un grafo no ponderado.

- **DFS**

  El algoritmo de búsqueda en profundidad, conocido como DFS (Depth-First Search), es una técnica para explorar grafos y árboles de manera más exhaustiva. A diferencia de la búsqueda en amplitud (BFS), DFS se sumerge tan profundamente como sea posible en una rama antes de retroceder y explorar otras ramas.

  El proceso comienza seleccionando un nodo de inicio y visitándolo. Luego, el algoritmo se mueve hacia uno de sus nodos vecinos no visitados más profundos y continúa explorando hasta llegar al final de esa rama. Cuando no hay más nodos por explorar en esa rama, el algoritmo retrocede al nodo anterior que aún tiene nodos no visitados y repite el proceso.

  Este enfoque de inmersión profunda y retroceso se repite hasta que se han explorado todas las ramas del grafo. Si hay nodos no visitados restantes, se selecciona uno como nuevo punto de partida y se repite el proceso.

  DFS es útil para diversas tareas, como la búsqueda de caminos, la detección de ciclos y la exploración completa de grafos. DFS es valioso cuando se busca explorar a fondo una rama antes de pasar a las demás. Este algoritmo se utiliza comúnmente en problemas que involucran la estructura y conectividad de grafos y árboles

- **Caminos de peso mínimo**

  - **Algoritmo de Dijkstra**

    El algoritmo de Dijkstra se utiliza para encontrar la ruta más corta entre un nodo de

inicio y todos los demás nodos en un grafo ponderado, donde las conexiones tienen pesos no negativos. En el proceso, se asigna una distancia inicial "infinita" a todos los nodos, excepto al nodo de inicio, al cual se le otorga una distancia de cero. Se crea un conjunto que incluye todos los nodos, considerándolos como no visitados.

A medida que avanza, el algoritmo selecciona el nodo no visitado con la distancia mínima conocida, inicialmente siendo el nodo de inicio. Luego, actualiza las distancias de los nodos vecinos en función de la distancia acumulada desde el nodo de inicio a través del nodo seleccionado. Si la nueva distancia es menor que la distancia previamente conocida para ese nodo vecino, se actualiza la distancia.

Este proceso se repite hasta que todos los nodos han sido visitados y se han determinado sus distancias más cortas desde el nodo de inicio. Al finalizar, no solo se obtienen estas distancias, sino que también es posible reconstruir los caminos más cortos.

El algoritmo de Dijkstra se destaca en situaciones que requieren rutas óptimas, como en redes de transporte o telecomunicaciones. Es esencial señalar que Dijkstra solo funciona adecuadamente en grafos con pesos no negativos; para situaciones con pesos negativos, se deben explorar enfoques alternativos como el algoritmo de Bellman-Ford.

- **Algoritmo de Floyd Warshall**

El algoritmo de Floyd-Warshall se utiliza para encontrar los caminos más cortos entre todos los pares de nodos en un grafo dirigido y ponderado. A diferencia de otros algoritmos, como Dijkstra, que se centran en encontrar caminos más cortos desde un nodo de origen, Floyd-Warshall aborda simultáneamente todos los pares de nodos en el grafo.

El proceso comienza inicializando una matriz de distancias, donde cada elemento (i, j) representa la distancia entre los nodos i y j. Se establecen distancias directas si hay una arista entre i y j, y se asigna infinito si no hay conexión directa.

Luego, el algoritmo procede a actualizar las distancias considerando nodos intermedios. Itera sobre todos los pares de nodos i y j, y para cada nodo intermedio k, verifica si la ruta desde i hasta j a través de k es más corta que la distancia actual entre i y j. Si es así, actualiza la distancia.

Este proceso de actualización se repite para cada nodo intermedio k, logrando que, al finalizar las iteraciones, la matriz de distancias contenga los caminos más cortos entre todos los pares de nodos.

El algoritmo de Floyd-Warshall es útil especialmente en grafos con aristas de peso negativo, siempre y cuando no existan ciclos de peso negativo accesibles desde el nodo de origen. Aunque es computacionalmente más costoso que algunos algoritmos, su capacidad para manejar diversas condiciones del grafo lo hace valioso en ciertos contextos.

- **Árbol de recubrimiento mínimo**

    - **Algoritmo de Prim**

      El algoritmo de Prim se utiliza para encontrar un árbol de expansión mínima en un grafo ponderado y conexo. En términos simples, busca el conjunto de aristas que conecta todos los nodos del grafo con el peso total más bajo posible.

      El proceso comienza eligiendo un nodo de inicio y construyendo gradualmente el árbol de expansión mínima. En cada paso, se selecciona la arista más pequeña que conecta un nodo en el conjunto actual con uno fuera del conjunto. Este nodo y la arista se agregan al conjunto y a la lista de aristas del árbol.

      Este proceso se repite hasta que todos los nodos están incluidos en el árbol de expansión mínima. Al final, el algoritmo asegura que la suma de los pesos de las aristas en el árbol sea la mínima posible.

      El algoritmo de Prim es eficaz, especialmente cuando se trata de grafos densos con muchas aristas. Se aplica en situaciones prácticas como la planificación de redes de comunicación, donde la optimización de costos es esencial.

    - **Algoritmo de Kruskal**

      El algoritmo de Kruskal es una herramienta de la teoría de grafos que busca el árbol de expansión mínima en un grafo conexo y ponderado. Su propósito es encontrar el conjunto mínimo de aristas que conecta todos los nodos del grafo con el peso total más bajo posible.

      El proceso inicia ordenando todas las aristas del grafo de menor a mayor según sus pesos. Luego, se inicializa un conjunto que contiene todos los nodos del grafo, pero sin aristas al principio. En cada paso, se selecciona la arista más ligera y se agrega al conjunto si su inclusión no crea un ciclo. Este proceso se repite hasta que todas las aristas han sido consideradas o hasta que todos los nodos están conectados.

      Al finalizar, el conjunto resultante de aristas forma el árbol de expansión mínima. El algoritmo de Kruskal asegura que la suma de los pesos de las aristas seleccionadas sea mínima y que no haya ciclos en el árbol resultante. Es especialmente efectivo en grafos dispersos, aquellos con pocas aristas.

FASE 3:

Con un sólido entendimiento del juego, sus reglas, la interacción con el usuario y su mecánica interna, ingresamos a la fase crítica de selección de algoritmos para el desarrollo de nuestro juego "Space Adventure". Nos centramos en encontrar soluciones eficientes que mejoren la experiencia del jugador, ya sea navegando manualmente entre planetas y estaciones o activando el piloto automático para explorar destinos más lejanos.

En el contexto de la navegación manual, donde el jugador busca explorar de manera consciente y eficiente, identificamos dos opciones destacadas: BFS (Breadth-First Search) y

DFS (Depth-First Search). Estos algoritmos ofrecen una primera validación para la ejecución del viaje y proporcionan una ruta hacia el destino, siempre y cuando se cumplan las condiciones previamente establecidas: viajar a no más de 3 planetas de distancia y contar con el combustible necesario según el peso de cada arista.

- DFS (Depth-First Search): Ofrece una exploración profunda, ideal para jugadores que desean sumergirse en la experiencia y explorar de manera detallada. Adecuado para validar y proporcionar rutas en la navegación manual.
- BFS (Breadth-First Search): Realiza una exploración en amplitud, proporcionando una ruta más directa entre los nodos. Opción eficiente para jugadores que prefieren una navegación más rápida y directa.

Para la funcionalidad de piloto automático, donde el jugador puede llegar a planetas desconocidos más lejanos, consideramos dos algoritmos fundamentales: Prim y Kruskal. Estos algoritmos evitan ciclos y garantizan una exploración eficiente y sin repeticiones.

- Prim:Construye un árbol de recubrimiento mínimo, perfecto para el piloto automático al seleccionar rutas eficientes. Minimiza el consumo de combustible, adaptándose a la limitación de distancia de 3 planetas.
- Kruskal: Selecciona aristas de menor peso, ofreciendo un enfoque similar al de Prim para el piloto automático. Garantiza la exploración óptima y la eficiencia en la utilización del combustible.

Entonces, con el análisis previo, planteamos las siguientes alternativas:

- *Alternativa 1. DFS - Prim*
- *Alternativa 2. BFS - Prim*
- *Alternativa 3. DFS - Kruskal*
- *Alternativa 4. BFS - Kruskal*

La elección final dependerá de factores como la complejidad del algoritmo y las preferencias de diseño. Es crucial destacar que todas las alternativas comparten características esenciales:

- Interfaz gráfica para una interacción cómoda del usuario.
- Distribución aleatoria de objetos de recolección y enemigos en los planetas.
- Generación aleatoria del mapa, incluyendo nodos, ubicaciones y aristas.

Con estas consideraciones, nos embarcamos en la fase de implementación, asegurando una experiencia de juego cautivadora y fluida para nuestros usuarios.

FASE 4:

A pesar de ser una opción eficiente para búsquedas rápidas y directas, el Breadth-First Search (BFS) se descarta para la navegación manual en el juego "Space Adventure" por las siguientes razones:

- Rutas poco intuitivas: BFS tiende a buscar la ruta más corta en términos de nodos visitados, lo que puede resultar en rutas que no sigan una lógica intuitiva en el espacio

del juego.
- Menor flexibilidad: En entornos con restricciones específicas, como la limitación de viajar a no más de 3 planetas de distancia, BFS puede no ser tan flexible para adaptarse a estas restricciones, ya que prioriza la amplitud sobre la profundidad.

Diseño preliminar

La implementación utilizará el algoritmo de búsqueda en profundidad (DFS) para la navegación manual del jugador entre planetas y estaciones, garantizando que se cumplan las restricciones de distancia y combustible.

Para la funcionalidad de piloto automático, se considerarán tanto Prim como Kruskal, dos algoritmos de árbol de expansión mínima, para construir rutas eficientes y sin ciclos.

- Navegación manual (DFS): El jugador selecciona manualmente el planeta de destino.Se realiza un recorrido DFS para validar la viabilidad del viaje y generar una ruta.Se verifica que la distancia no supere las 3 unidades y que haya suficiente combustible según el peso de las aristas.
- Piloto automático (Prim/Kruskal): El sistema evalúa la cantidad de combustible disponible y la distancia máxima permitida. Utilizando Prim o Kruskal, se genera un árbol de expansión mínima que conecta todos los planetas.Se selecciona un planeta de destino más lejano dentro de las restricciones establecidas.
- Interfaz gráfica:** Se mostrará un mapa interactivo que refleja la disposición aleatoria de los planetas y las conexiones entre ellos. La interfaz permitirá al jugador realizar selecciones manuales o activar el piloto automático.
- Objetos y enemigos:Los objetos de recolección y los enemigos se distribuirán aleatoriamente en los planetas para garantizar la rejugabilidad.
- Generación del mapa: Tanto los nodos (planetas y estaciones) como las aristas (conexiones) se generarán de manera aleatoria para proporcionar una experiencia única en cada partida.

Este diseño preliminar permite la flexibilidad de implementar tanto Prim como Kruskal en la fase de piloto automático, lo que permite ajustar la elección según los criterios establecidos en la siguiente fase.

.

FASE 5:

Criterios de Evaluación:

- Eficiencia del Algoritmo: Este criterio evalúa la rapidez y eficacia con la que el algoritmo realiza sus cálculos. En el contexto de la navegación en el juego, una mayor eficiencia significa tiempos de respuesta más rápidos, lo que contribuye a una experiencia de juego más fluida. La escala de evaluación va desde "Muy baja" hasta "Muy alta", reflejando la velocidad y eficiencia del algoritmo.
- Adaptabilidad a Restricciones del Juego: Evalúa en qué medida la alternativa se adapta a las restricciones y reglas específicas del juego, como la limitación de viajar a no más de 3 planetas de distancia y la necesidad de combustible según el peso de las aristas. Una

mayor puntuación indica una mejor adaptabilidad a estas restricciones.

- Intuitividad de Rutas: Este criterio mide la claridad y facilidad con la que el algoritmo presenta las rutas al jugador. Una ruta intuitiva facilita la comprensión y la toma de decisiones por parte del jugador. La escala refleja desde rutas poco claras hasta rutas muy intuitivas.
- Experiencia del Usuario: Evalúa la experiencia general del usuario al interactuar con el juego, considerando aspectos como la facilidad de uso, la satisfacción del usuario y la inmersión en el juego. Una puntuación más alta indica una experiencia de usuario más positiva.

	Eficiencia del algoritmo	Adaptabilidad	Intuitividad	Experiencia de usuario	Total
DFS-Prim	4	5	4	5	14
DFS-Kruskal	3	3	4	4	12

Según la evaluación de los criterios establecidos, la alternativa de Prim-DFS obtuvo un total de 14 puntos, superando a Kruskal-DFS que obtuvo 12 puntos. Por lo tanto, la elección recomendada para la implementación en el juego sería Prim-DFS debido a su mayor eficiencia, adaptabilidad a restricciones específicas del juego, intuitividad de rutas y experiencia del usuario.

Bibliografía

Algoritmo de Kruskal para encontrar el árbol de expansión mínimo. (s/f). Techiedelight.com. Recuperado el 25 de noviembre de 2023, de https://www.techiedelight.com/es/kruskals-algorithm-for-finding-minimum-spanning-tree/

Barrat, A., Barthélemy, M., Pastor-Satorras, R., & Vespignani, A. (2004). The architecture of complex weighted networks. Proceedings of the National Academy of Sciences of the United States of America, 101(11), 3747–3752. https://doi.org/10.1073/pnas.0400087101

Específicos, O. (s/f). Tema: Recorrido de Grafos. Aplicaciones de Grafos. Edu.sv. Recuperado el 25 de noviembre de 2023, de https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-9.pdf

Floyd Warshall algorithm. (2012, junio 7). GeeksforGeeks. https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

How to find Shortest Paths from Source to all Vertices using Dijkstra s Algorithm. (2012, noviembre 25). GeeksforGeeks. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Lista de Adyacencia. (2013, junio 5). koketxt. https://koketxt.wordpress.com/unidad-iii/representacion-de-grafos/lista-de-adyacencia/

Matrices y grafos. (s/f). Calculo.cc. Recuperado el 25 de noviembre de 2023, de https://calculo.cc/temas/temas_algebra/matriz/teoria/matriz_grafo.html

Navone, E. C. (2022, octubre 24). Algoritmo de la ruta más corta de Dijkstra - Introducción gráfica y detallada. freecodecamp.org. https://www.freecodecamp.org/espanol/news/algoritmo-de-la-ruta-mas-corta-de-dijkstra-introduccion-grafica/

Representar grafos (artículo). (s/f). Khan Academy. Recuperado el 25 de noviembre de 2023, de https://es.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs

Teoría de grafos. (s/f). Ecured.cu. Recuperado el 25 de noviembre de 2023, de https://www.ecured.cu/Teoría_de_grafos