JUEGO DE CITAS - SPARK

COMPUTACIÓN Y ESTRUCTURAS DISCRETAS

JUANJO CONTRERAS

DAMY VILLEGAS

JUAN ESTEBAN ERAZO

21/10/2023

**METODO DEL DISEÑO DE INGENIERIA**

DESCRIPCIÓN DEL CONTEXTO PROBLEMÁTICO:

El contexto del problema parece centrarse en el desarrollo de un juego de mesa basado en grafos con elementos desafiantes y entretenidos. Se requiere crear un juego que utilice al menos dos algoritmos de grafos (como BFS, DFS, Dijkstra, Floyd-Warshall, Prim, Kruskal) en un tablero que represente relaciones entre jugadores, elementos del juego o relaciones dentro del mundo del juego.

IDENTIFICACIÓN Y DEFINICIÓN DEL PROBLEMA:

El problema principal es el desarrollo de un juego basado en un grafo con un mínimo de 50 vértices y 50 aristas. Este juego debe ser desafiante y entretenido para los jugadores, permitiendo movimientos estratégicos donde las aristas y vértices del grafo representen relaciones entre jugadores, elementos del juego o características del mundo del juego.

ESPECIFICACIÓN DE REQUERIMIENTOS FUNCIONALES:

Desarrollo de Grafos (Dos Versiones):

Implementar dos versiones del grafo desde su TAD hasta las pruebas unitarias automáticas.

MÉTODO DE INGENIERÍA PARA LA SOLUCIÓN:

Documentar cada fase del método de ingeniería, desde el análisis hasta la implementación.

DISEÑO DEL JUEGO:

El tablero debe representar un mundo desafiante con diferentes tipos de casillas y relaciones entre ellas.

Las aristas pueden representar relaciones entre jugadores, elementos del juego o características del mundo del juego.

DESARROLLO DE LAS IDEAS DEL JUEGO:

* UTILIZACIÓN DE LAS ARISTAS:

Representación de relaciones entre jugadores, elementos del juego o características del mundo del juego.

* POSIBLES EJEMPLOS DE USO DE ARISTAS:

1. Amistades, alianzas, enemistades entre jugadores.
2. Relaciones comerciales, románticas o de poder entre jugadores y elementos del juego.
3. Relaciones entre ciudades, personajes u objetos dentro del mundo del juego.

SÍNTESIS:

El desafío consiste en crear un juego innovador y desafiante basado en grafos, donde los jugadores puedan mover sus piezas por un tablero representado por un grafo. Las relaciones entre los nodos y aristas del grafo pueden representar varios elementos del mundo del juego, como relaciones entre jugadores, elementos o características especiales del mundo del juego.

DEFINICION DE NUESTRO JUEGO A IMPLEMENTAR

Nuestro proyecto por desarrollar siguiendo todas las reglas del enunciado, se relaciona con diferentes aplicaciones de citas, las cuales son creadas para conocer a diferentes personas que tengan gustos en común, nuestra idea a plantear es la siguiente.

IDEA

El juego se lleva a cabo mediante una interfaz gráfica que representa un tablero, simulando una aplicación de citas. El jugador participante tiene la oportunidad de ingresar sus datos personales como nombre, edad, intereses y hobbies. Estos datos son fundamentales para comparar al jugador con una variedad de usuarios registrados en el sistema.

La mecánica del juego se basa en buscar la compatibilidad del jugador con otros usuarios dentro de la aplicación. Esta compatibilidad se determina mediante la comparación de los datos proporcionados por el jugador con los perfiles de otros usuarios registrados en la plataforma.

Se utilizan algoritmos y criterios predefinidos para analizar la información ingresada por el jugador y encontrar coincidencias o similitudes con los datos de otros usuarios. Estos criterios pueden incluir intereses en común, edades cercanas, hobbies compartidos o cualquier otro factor relevante para establecer una posible compatibilidad entre dos personas.

Una vez que se realiza esta comparación, se presenta al jugador una lista de perfiles de otros usuarios que tienen más posibilidades de ser compatibles con base en la información proporcionada. De esta manera, el jugador puede explorar estos perfiles y tomar decisiones dentro del juego para encontrar su "pareja" o persona más afín de entre las opciones presentadas.

Este enfoque de juego a través de una aplicación de citas busca ofrecer una experiencia interactiva y entretenida, donde el jugador tiene la oportunidad de encontrar personas afines basándose en sus intereses, hobbies y otros datos personales compartidos.

OBJETIVO

El jugador a través de una experiencia interactiva y atractiva tiene la posibilidad de descubrir qué usuarios dentro del juego podrían tener más afinidad o similitudes con base en los datos proporcionados.

ALGORITMOS

Con la implementación de nuestros algoritmos de grafos sirven para que los jugadores encuentren a su persona mas compatible y asi generar conexiones.

USO DEL ALGORITMO DIJKSTRA

A usar dentro del programa:

* El uso de la función shortestPathBasedOnInterests que calcula la distancia basada en la similitud de intereses entre dos perfiles podría tener varios propósitos dentro de tu aplicación de citas:
* Recomendaciones de coincidencia: Puedes utilizar esta función para sugerir conexiones más fuertes entre usuarios. Por ejemplo, podrías mostrar perfiles con mayor similitud de intereses como posibles coincidencias destacadas o destacar la compatibilidad entre dos usuarios.

Posibles a usar dentro del programa:

* Filtrado y búsqueda avanzada: Podrías permitir a los usuarios filtrar resultados de búsqueda o sugerencias de coincidencias basadas en la similitud de intereses. Esto les daría la opción de buscar perfiles que tengan más probabilidades de tener intereses compartidos.
* Mejora de algoritmos de recomendación: Si estás implementando algoritmos de recomendación más complejos en tu aplicación, esta función podría ser útil como un componente en la evaluación de la compatibilidad entre usuarios para mejorar la precisión de las recomendaciones.

DETALLER ADICIONALES

1. Numero de Jugadores: 1 jugador.
2. Duración del Juego: La duración del juego depende del usuario.
3. Edad Recomendada: El juego es para personas de 18 en adelante
4. Mensajes enviados por el usuario: El usuario tiene la posibilidad de enviar mensajes y organizar citas con las personas que logre mayor compatibilidad.

ESPECIFICACION DE REQUERIMIENTOS

1. Tablero y Grafos

* Creación del Tablero: Diseño de un tablero gráfico que simule una app de citas o un juego de relaciones entre personas con al menos 50 vértices (usuarios) y 50 aristas (conexiones entre los usuarios).
* Implementación del Grafo: Desarrollo de una estructura de grafo que represente las relaciones entre los usuarios. Los vértices representarán a los usuarios registrados, y las aristas las conexiones entre ellos.

2. Interfaz Gráfica y Movimientos

* Desarrollo de Interfaz Gráfica: Creación de una interfaz visual que muestre el tablero y permita a los jugadores interactuar con él.
* Movimiento de Jugadores: Implementación de la interfaz para que pueda registrar usuarios y asi se comparen los gustos y preferencias que este haya ingresado.

4. Mensajes y Organizaciones de citas:

* Los jugadores tienen la opción de poder enviar mensajes y organizar citas con las personas que haya tenido mayor compatibilidad.

5. Algoritmos de Grafos:

* Implementación de Algoritmos: Desarrollo de al menos tres algoritmos de grafos (BFS, DFS, Caminos de Peso Mínimo) para que los jugadores los utilicen estratégicamente durante el juego.

7. Jugabilidad y Testing:

* Pruebas y Depuración: Realización de pruebas exhaustivas para garantizar la jugabilidad, corrección y fluidez del juego.
* Iteraciones y Mejoras: Realización de ajustes y mejoras basadas en los resultados de las pruebas y el feedback de los jugadores beta.

RECOPILACION DE LA INFORMACION NECESARIA

Para el desarrollo de este proyecto se necesita de una recopilación de información:

1. Algoritmos de Grafos

Los algoritmos de grafos son un conjunto de instrucciones que recorren (visitan los nodos de) un grafo. Algunos algoritmos son usados para hallar un nodo específico o el camino entre dos nodos dados. Los grafos son estructuras de datos muy útiles que se usan para modelar varios problemas. Estos algoritmos tienen aplicaciones directas en sitios de redes sociales, modelado de máquinas de estado y más. Algunos de los algoritmos de grafos más comunes son:

* Búsqueda en Amplitud o Anchura (Breadth First Search, BFS):

Búsqueda en Amplitud o Anchura es uno de los algoritmos de grafos más sencillo. Recorre el grafo al primero comprobar el nodo actual y luego expandirlo al agregar sus sucesores al siguiente nivel. El proceso se repite para todos los nodos del nivel actual antes de moverse al siguiente. Si se encuentra la solución, la búsqueda termina.

* Búsqueda en Profundidad (Depth First Search, DFS):

La Búsqueda en Profundidad es uno de los algoritmos de grafos más sencillos. Recorre el grafo revisando primero el nodo actual y moviéndose después a uno de sus sucesores para repetir el proceso. Si el nodo actual no tiene sucesor a revisar, regresamos a su predecesor y el proceso continúa (moviéndose a otro sucesor). Si la solución es encontrada, la búsqueda termina.

* Dijkstra

El Algoritmo de Dijkstra es un algoritmo de grafo presentado por E. W. Dijkstra. Encuentra el camino más corto de origen único en un grafo con aristas no negativas.

Creamos dos arreglos: visitado y distancia, que registran si un vértice es visitado y cuál es la mínima distancia desde el vértice origen, respectivamente. Inicialmente, el arreglo visitado se asigna como falso y distancia como infinito.

Partimos del vértice origen. Dejemos que el vértice actual sea u y sus vértices adyacentes sean v. Ahora, por cada v que es adyacente a u, la distancia se actualiza si no ha sido visitado antes y la distancia desde u es menor que su distancia actual. Luego seleccionamos el siguiente vértice con la menor distancia y que no haya sido visitado.

La Cola de Prioridad se usa a menudo para cumplir este último requerimiento en el menor tiempo posible. Abajo se ve una implementación de la misma idea usando la cola de prioridad en Java.

* Algoritmo de Floyd-Warshall:

El algoritmo de Floyd Warshall es un gran algoritmo para encontrar la distancia más corta entre todos los vértices del grafo. Tiene un algoritmo muy conciso y complejidad del tiempo O(V^3) (donde V es el número de vértices). Puede ser usado con pesos negativos, aunque los ciclos de pesos negativos no deben estar presentes en el grafo.

Fuente: <https://www.freecodecamp.org/espanol/news/explicacion-de-algoritmos-y-estructuras-de-datos-de-grafos-con-ejemplos-en-java-y-c/#:~:text=%C2%BFQu%C3%A9%20es%20un%20Algoritmo%20de,camino%20entre%20dos%20nodos%20dados>.

1. Grafos, Vértices y Aristas

GRAFOS

Un grafo consiste en un conjunto finito de puntos llamados vértices y un conjunto finito de aristas, cada una de las cuales conecta dos vértices.

Se dice que dos vértices son adyacentes, si están conectados por una arista.

Matriz de adyacencia: Dado un grafo G que tiene n vértices, su matriz de adyacencia es la matriz A(G) de tamaño n × n definida por:

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja

La anterior definición es apropiada solamente para los grafos simples. Un grafo es simple si no existen múltiples aristas que conectan a una pareja de vértices.

Fuente: <https://ciencias.medellin.unal.edu.co/cursos/algebra-lineal/clases/8-clases/40-clase-2-parte3.html#:~:text=Un%20grafo%20consiste%20de%20un,est%C3%A1n%20conectados%20por%20una%20arista>.

VERTICES

El grado de un vértice en un grafo es el número de aristas incidentes a él. Un vértice aislado es un vértice con grado cero; esto es, un vértice que no es punto final de ninguna arista. Un vértice hoja es un vértice con grado uno. En un grafo dirigido, se puede distinguir entre grado de salida ("outdegree", número de aristas que salen del vértice) y grado de entrada ("indegree", número de aristas que llegan al vértice); un vértice fuente es un vértice con grado de entrada cero, mientras que un sumidero es un vértice con grado de salida cero.

Un vértice de corte es un vértice que al removerlo desconecta al grafo restante. Un conjunto independiente es un conjunto de vértices tal que ninguno es adyacente a otro, y una cobertura de vértices es un conjunto de vértices que incluye los puntos finales de cada arista en un grafo.

La vecindad de un vértice x, denotado como N(x) está dado por todos los vértices adyacentes a x.

Fuente: <https://es.wikipedia.org/wiki/V%C3%A9rtice_(teor%C3%ADa_de_grafos)#:~:text=En%20teor%C3%ADa%20de%20grafos%2C%20un,que%20est%C3%A1n%20formados%20los%20grafos>.

ARISTAS

Son las líneas que unen los vértices de un grafo. Hay diferentes tipos de aristas:

* Aristas adyacentes: Dos aristas son adyacentes si convergen en el mismo vértice.
* Aristas paralelas: Dos aristas son paralelas si los vértices iniciales y finales son el mismo vértice
* Aristas cíclicas: Aristas que parten de un vértice para entrar en el mismo.

Términos importantes

* Cruce: Punto donde dos aristas se cruzan.
* Vértices: Los vértices son los elementos que forman un grafo. Cada uno lleva asociada una valencia característica según la situación, que se corresponde con la cantidad de aristas que confluyen en dicho vértice.
* Camino: Se denomina camino a un conjunto de vértices interconectados por aristas. Dos vértices están conectados si hay un camino entre ellos.

Fuente: <https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos#:~:text=Aristas%3A%20Son%20las%20l%C3%ADneas%20que,convergen%20en%20el%20mismo%20v%C3%A9rtice.&text=Aristas%20c%C3%ADclicas%3A%20Aristas%20que%20parten,para%20entrar%20en%20el%20mismo>.

1. Implementación de una Interfaz Grafica

Interfaz Gráfica de Usuario, interfaz de usuario o GUI (Graphical User Interface): es el entorno de objetos gráficos disponibles para un usuario en el marco de una aplicación o sistema operativo. El sistema operativo MS-Dos se basaba en intérpretes de comando (escritura de instrucciones por consola) pero Windows se basa en una interfaz gráfica de usuario (su entorno de escritorio), Linux en otra y Macintosh en otra.

Herramientas para crear Interfaces gráficas de usuario en Java. Hacemos referencia principalmente a los paquetes (packages) del API de Java swing y awt (Abstract Windowing Toolkit). Las clases de estos paquetes permiten crear interfaces gráficas de usuario basadas en ventanas estilo “Windows” para nuestras aplicaciones.

Fuente: <https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=644:concepto-o-definicion-de-interface-java-ique-es-una-interface-tipos-de-interfaz-ejemplo-practico-cu00677b&catid=68&Itemid=188#:~:text=c)%20Interfaz%20Gr%C3%A1fica%20de%20Usuario,una%20aplicaci%C3%B3n%20o%20sistema%20operativo>.

1. Listas de Adyacencia

Una lista de adyacencia es una estructura de datos bidimensional que permite búsquedas e inserciones eficientes y, a su vez, es óptima en el uso de memoria. Uno de sus principales usos es en la representación de relaciones binarias (o grafos) y la representación de listas con prioridad.

En particular, la estructura de datos lista de adyacencia ofrece las siguientes operaciones:

* Init (): crea una lista de adyacencia vacía (complejidad temporal O (1)).
* Eq (other): determina si la lista de adyacencia es igual a other (complejidad temporal O(n), en donde n es la cantidad de elementos en la lista de adyacencia).
* str (): representación en cadena de la lista de adyacencia (complejidad temporal O(n), en donde n es la cantidad de elementos en la lista de adyacencia).
* Insert (x, y): inserta el elemento y con prioridad x (complejidad temporal O (log2 n), en donde n es la cantidad de elementos en la lista de adyacencia).
* Contains (x, y): retorna True si el elemento y con prioridad x está en la lista de adyacencia; False de lo contrario (complejidad temporal O (log2 n), en donde n es la cantidad de elementos en la lista de adyacencia).
* Remove (x, y): elimina el elemento y con prioridad x de la lista de adyacencia, si este existe (complejidad temporal O (log2 n), en donde n es la cantidad de elementos en la lista de adyacencia).

Fuente: <https://www.camilorocha.info/teaching/pimo/2013-1/laboratorios/laboratorio8listasdeadyacencia>

1. Matriz de Adyacencia

Es una matriz booleana que representa las conexiones entre pares de vértices. La matriz de adyacencia de un grafo es simétrica. Si un vértice es aislado entonces la correspondiente fila (columna) está compuesta sólo por ceros. Si el grafo es simple entonces la matriz de adyacencia contiene solo ceros y unos (matriz binaria) y la diagonal está compuesta sólo por ceros.

La matriz de adyacencia de un dígrafo no es simétrica. Es una matriz binaria. El número de unos que aparecen en una fila es igual al grado de salida del correspondiente vértice y el número de unos que aparecen en una determinada columna es igual al grado de entrada del correspondiente vértice.

Algoritmo para construir la matriz de adyacencia de un grafo.

* Paso 1: Se asignan un orden arbitrario a los vértices.
* Paso 2: Se construye una matriz de dimensión n\*n, cardinalidad (# vértices) por cardinalidad (# vértices).
* Paso 3: En la posición I j-ésima se coloca 1 si el vértice i es adyacente al vértice j; 0 en el otro caso

Ejemplo:

Imagen que contiene Diagrama

Descripción generada automáticamente

Fuente: <https://www.utm.mx/~mgarcia/ED4(Grafos).pdf>

1. Pruebas Unitarias Automáticas

Las pruebas automatizadas consisten en la aplicación de herramientas de software para automatizar el proceso manual de revisión y validación de un producto de software que lleva a cabo una persona. Ahora, la mayoría de los proyectos de software ágiles y de DevOps modernos incluyen pruebas automatizadas desde el principio; sin embargo, para apreciar plenamente el valor de dichas pruebas, hay que saber cómo era la vida antes de que se adoptaran de forma generalizada.

¿Por qué son importantes?

La entrega continua (CD) consiste en publicar versiones de código nuevas lo más rápido posible para los clientes. Las pruebas automatizadas resultan fundamentales para alcanzar ese objetivo. No hay forma de automatizar dicha publicación si hay un paso manual que requiere mucho tiempo en el proceso de publicación.

La CD forma parte de un proceso de implementación mayor. Es sucesora de la integración continua (CI) y depende de ella. La CI es totalmente responsable de ejecutar pruebas automatizadas ante cualquier cambio de código nuevo y de verificar que dichos cambios no afectan a la integridad de las funciones establecidas ni introducen errores nuevos. La CD se activa una vez que el paso de integración continua supera el plan de pruebas automatizado.

Esta relación entre las pruebas automatizadas, la CI y la CD aporta numerosas ventajas a los equipos de software que trabajan a gran velocidad. Las pruebas automatizadas garantizan la calidad en todas las fases del desarrollo, ya que aseguran que las confirmaciones nuevas no introducen ningún error, por lo que el software sigue estando listo para implementarse en todo momento.

Fuente: <https://www.atlassian.com/es/continuous-delivery/software-testing/automated-testing#:~:text=Las%20pruebas%20automatizadas%20consisten%20en,lleva%20a%20cabo%20una%20persona>.

1. Tablas Hash

Una tabla hash, matriz asociativa, hashing, mapa hash, tabla de dispersión o tabla fragmentada es una estructura de datos que implementa el tipo de dato abstracto llamado diccionario (tipo de dato abstracto). Esta asocia llaves o claves con valores.​ La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos (teléfono y dirección, por ejemplo) almacenados a partir de una clave generada (usando el nombre o número de cuenta, por ejemplo). Funciona transformando la clave con una función hash en un hash, un número que identifica la posición (casilla o cubeta) donde la tabla hash localiza el valor deseado.

Las tablas hash se suelen implementar sobre vectores de una dimensión, aunque se pueden hacer implementaciones multidimensionales basadas en varias claves. Como en el caso de los arrays, las tablas hash proveen tiempo constante de búsqueda promedio sin importar el número de elementos en la tabla. Sin embargo, en casos particularmente malos el tiempo de búsqueda puede llegar a O(n), es decir, en función del número de elementos.

Comparada con otras estructuras de arrays asociadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información.

Las tablas hash almacenan la información en posiciones pseudo-aleatorias, así que el acceso ordenado a su contenido es bastante lento. Otras estructuras como árboles binarios auto-balanceables tienen un tiempo promedio de búsqueda mayor (tiempo de búsqueda O(log n)), pero la información está ordenada en todo momento.

Fuente: <https://es.wikipedia.org/wiki/Tabla_hash#:~:text=Comparada%20con%20otras%20estructuras%20de,su%20contenido%20es%20bastante%20lento>.

Tablas Hash en Java

La clase Hashtable es la implementación de una estructura de datos de tabla hash. Esta colección se creó antes que Java Collection Framework, pero luego se incluyó en ella. Como todas las colecciones "tempranas" (desde Java 1.0), una tabla hash está sincronizada (casi todos los métodos están marcados como sincronizados). Debido a este factor, la tabla hash tiene importantes problemas de rendimiento. Por lo tanto, a partir de Java 1.2, en la mayoría de los casos se recomienda utilizar otras implementaciones de la interfaz Map debido a su falta de sincronización. Por lo general , HashMap es el reemplazo más apropiado. Entonces clase Hashtable<K,V>consta de claves y valores. Almacena claves según el principio de hashing. Los pares clave-valor se almacenan en "cubos". Los cubos juntos construyen una "tabla", una especie de matriz interna. Hashtable usa el código hash de la clave para determinar un depósito en el que debe asignarse el par clave/valor. La función hash permite obtener la ubicación del depósito a partir del código hash de Key. Esta función devuelve un número entero para un objeto. Como dijimos anteriormente, dos objetos iguales tienen el mismo código hash, mientras que dos objetos desiguales pueden no tener siempre códigos hash diferentes. Diferentes objetos, colocados en una tabla hash pueden tener el mismo código hash. Para resolver este problema (colisión), se utiliza una matriz de listas en la tabla hash. Los pares asignados a un solo depósito se almacenan en una lista y esta referencia de lista se almacena en el índice de matriz.

Fuente: <https://codegym.cc/es/groups/posts/es.218.tabla-hash-de-java>

BUSQUEDA DE SOLUCIONES CREATIVAS

1. Tablero y Grafos

Creación del Tablero:

* Tablero Tridimensional: Define cómo los vértices y las aristas del grafo se representarán en el tablero. Podrías considerar un enfoque en tres dimensiones, donde los vértices se coloquen en diferentes niveles o capas del tablero, y las aristas se muestren como conexiones entre estos vértices en esas capas.

Implementa una representación visual en 3D del tablero utilizando bibliotecas gráficas disponibles en Java, como JavaFX o OpenGL. Esta representación tridimensional permitirá al jugador visualizar los vértices y aristas del grafo de manera más interactiva.

* Tablero Modular: Define diferentes tipos de casillas o nodos que representarán a los usuarios o elementos del juego. Las aristas podrían representar relaciones entre estas casillas, como amistades, intereses compartidos o cualquier relación significativa para el juego.

Implementación del Grafo:

* Grafo Dinámico: Se implementaría una estructura de datos como un grafo dirigido o no dirigido, donde los nodos representen a los usuarios y las aristas a las relaciones entre ellos.

Para lograr esto, se utiliza la implementación de un grafo mediante la definición de clases como Nodo y Grafo. La clase Nodo contiene atributos que representen las características de los usuarios (nombre, edad, intereses, etc.), mientras que la clase Grafo podría contener métodos para agregar nodos, agregar relaciones entre nodos, así como implementaciones de algoritmos de grafos como BFS, DFS o Dijkstra para realizar comparaciones y encontrar la compatibilidad entre los usuarios.

* Grafo Interactivo: Se emplea una interfaz gráfica en la que los usuarios puedan interactuar visualmente con los nodos y las relaciones. Puedes utilizar bibliotecas gráficas como JavaFX o Swing para crear la interfaz gráfica y representar el grafo como un tablero visual donde los usuarios puedan agregar sus datos, ver sus conexiones y enviar mensajes o organizar citas con aquellos que tengan mayor compatibilidad.

1. Interfaz Gráfica y Movimientos

Desarrollo de la Interfaz Gráfica:

* Realidad Aumentada: Crear una interfaz de realidad aumentada que integre el tablero físico con elementos digitales, permitiendo a los jugadores interactuar con el tablero de forma más animada, esto puede realizarse utilizando bibliotecas específicas para manipulas la cámara y los gráficos en tiempo real, como Java3D.
* Personalización de Interfaz: La interfaz gráfica reflejará un tablero interactivo donde los jugadores podrán explorar perfiles de otros usuarios. Utilizará al menos dos algoritmos de grafos (como BFS, DFS, Dijkstra o Floyd-Warshall) para gestionar las relaciones entre los jugadores.

Se desarrollará un tablero con al menos 50 vértices (usuarios) y 50 aristas (conexiones entre ellos). Cada vértice representará a un usuario, y las aristas serán las relaciones entre estos usuarios.

La interfaz gráfica permitirá a los usuarios registrarse y comparar sus gustos y preferencias. Los jugadores podrán enviar mensajes y organizar citas con aquellos que tengan mayor compatibilidad basada en los algoritmos de grafos implementados.

Movimiento de los Jugadores:

* Movimientos Especiales: Implementar movimientos especiales adicionales que no sean solo ingresar información personal, como por ejemplo enviar mensajes y organizar citas con diferentes jugadores.
* Control de Movimiento: Permitir a los jugadores ingresar su propia información de forma manual y divertida.

DESCARTE DE IDEAS NO VIABLES

* Tablero Tridimensional:

Utiliza bibliotecas gráficas como Java3D o Unity (si fuera una opción) para crear una representación visual en 3D del tablero.

Asigna niveles o capas a los vértices del grafo para representar relaciones jerárquicas o diferentes conexiones.

Muestra las aristas como conexiones entre estos vértices en las capas tridimensionales.

* Grafo Dinámico:

Implementa un grafo dirigido o no dirigido donde los nodos representan usuarios y las aristas sus relaciones.

Cada nodo contendrá atributos de usuario (nombre, edad, intereses) y métodos para agregar relaciones.

Utiliza algoritmos de grafos para comparar datos de usuarios y encontrar compatibilidades.

* Interfaz Gráfica en Realidad Aumentada:

Manipula la cámara y los gráficos en tiempo real para integrar elementos digitales en el tablero físico.

Los jugadores podrán interactuar con la interfaz, registrarse, explorar perfiles y enviar mensajes dentro del mundo de realidad aumentada.

PERSONALIZACIÓN DE LA INTERFAZ Y MOVIMIENTOS:

* Interfaz Personalizada:

La interfaz mostrará un tablero interactivo con nodos (usuarios) y aristas (relaciones).

Permitirá a los jugadores ingresar información, explorar perfiles y enviar mensajes a aquellos con alta compatibilidad.

Utilizará algoritmos de grafos (BFS, DFS) para gestionar las relaciones entre jugadores y personalizar las recomendaciones.

* Movimientos Especiales:

Implementa acciones adicionales como la posibilidad de organizar citas, realizar búsquedas avanzadas basadas en preferencias y gustos comunes.

Permite a los usuarios interactuar con los nodos (usuarios) y aristas (relaciones) para activar estas funciones especiales.

DESCARTE DE IDEAS NO VIABLES:

* Descartamos las Tecnologías No Adaptables:

Descartamos el uso de ciertas bibliotecas gráficas debido a limitaciones de soporte o incompatibilidad con el concepto de realidad aumentada.

Se opta por no usar tecnologías demasiado complejas que podrían afectar la usabilidad del juego.

* Eliminación de Funciones Redundantes:

Descartamos funciones que podrían ser redundantes o confusas para los usuarios, manteniendo solo aquellas que mejoran la experiencia de juego y la interacción entre los usuarios.

DISEÑOS PRELIMINARES

TABLERO TRIDIMENSIONAL:

* Se utilizará la biblioteca Java3D para crear una representación visual en 3D del tablero.
* Los vértices se dispondrán en diferentes niveles dentro del espacio tridimensional para representar usuarios.
* Las aristas se mostrarán como conexiones visuales entre estos usuarios en los niveles correspondientes.

GRAFO DINÁMICO:

* Se implementará una clase Usuario con atributos como nombre, edad, intereses y métodos para manejar relaciones con otros usuarios.
* La clase Grafo contendrá la lógica para crear y gestionar conexiones entre usuarios.
* Algoritmos como BFS y DFS se utilizarán para comparar usuarios y encontrar compatibilidades basadas en sus atributos.

INTERFAZ GRÁFICA EN REALIDAD AUMENTADA:

* Se trabajará con la cámara y elementos gráficos en tiempo real para fusionar los elementos digitales en el tablero físico.
* Los jugadores podrán interactuar con esta interfaz para ingresar información, explorar perfiles y enviar mensajes dentro del entorno de realidad aumentada.
* Diseño Preliminar para Personalización de la Interfaz y Movimientos:

INTERFAZ PERSONALIZADA:

* La interfaz mostrará un tablero interactivo donde cada nodo representará a un usuario con información visible (nombre, intereses).
* Al seleccionar un nodo, se visualizarán las conexiones (aristas) y la información del usuario en una ventana emergente.
* Los jugadores podrán ingresar su información personal y explorar los perfiles de otros usuarios.

MOVIMIENTOS ESPECIALES:

* Se implementarán acciones adicionales, como la posibilidad de enviar mensajes a usuarios seleccionados.
* Habrá un sistema de búsqueda avanzada basado en los intereses compartidos entre los usuarios.
* Los jugadores podrán activar citas con otros usuarios con alta compatibilidad, llevando la interacción a un nivel más personal.
* Estos diseños preliminares abordan los aspectos clave de la implementación de la interfaz gráfica en realidad aumentada y la personalización de la experiencia del usuario en el juego.

EVALUACION Y SELECCIÓN DE LA MEJOR SOLUCION

* Experiencia del Usuario (UX):

Escalas:

1 a 5: Pobre a Excelente

Explicación: Evalúa la facilidad de uso, interactividad y satisfacción del usuario con la interfaz.

* Viabilidad Técnica:

Escalas:

1 a 5: Baja a Alta

Explicación: Evalúa la posibilidad técnica y la complejidad de implementación de cada idea.

* Interactividad de la Realidad Aumentada:

Escalas:

1 a 5: Baja a Alta

Explicación: Evalúa la efectividad y la inmersión proporcionada por la realidad aumentada en la interfaz.

* Funcionalidades Adicionales:

Escalas:

1 a 5: Limitadas a Abundantes

Explicación: Evalúa la presencia de funciones complementarias útiles más allá de la interfaz principal.

EVALUACIÓN DE LAS IDEAS

* Interfaz Gráfica en Realidad Aumentada:

1. Experiencia del Usuario: 4
2. Viabilidad Técnica: 3
3. Interactividad de la Realidad Aumentada: 5
4. Funcionalidades Adicionales: 4

* Personalización de la Interfaz y Movimientos:

1. Experiencia del Usuario: 5
2. Viabilidad Técnica: 4
3. Interactividad de la Realidad Aumentada: 3
4. Funcionalidades Adicionales: 5

TOTALIZACIÓN DE LA EVALUACIÓN:

* Interfaz Gráfica en Realidad Aumentada:

Puntaje total: 16

* Personalización de la Interfaz y Movimientos:

Puntaje total: 17

Según la evaluación basada en los criterios definidos, la "Personalización de la Interfaz y Movimientos" obtiene un puntaje total ligeramente más alto, lo que la convierte en la idea más viable y alineada con los criterios de evaluación establecidos. Por tanto, sería la idea seleccionada para la implementación, ya que tiene un puntaje más alto en general y parece ser más adecuada en términos de experiencia del usuario, viabilidad técnica y funcionalidades adicionales.