Método de la ingeniería

Paso 1. Identificación del Problema

Contexto: "Venganza Romana" es un emocionante juego de estrategia basado en un grafo no dirigido con más de 50 nodos y aristas que representa una parte de la Europa medieval. El jugador comienza en la histórica ciudad de Sicilia con el objetivo final de conquistar la base vikinga. Para lograrlo, deberá conquistar diferentes reinos, ciudades y pueblos, enfrentándose a desafíos estratégicos en cada territorio.

El mapa está organizado en niveles, con Sicilia como punto de partida. El jugador avanza a través de pueblos interconectados, cada uno con rutas que conducen a niveles superiores Cada territorio tiene un peso asociado, indicando la dificultad de conquista. El jugador debe tomar decisiones estratégicas para maximizar recursos y minimizar pérdidas de soldados.

Síntomas y necesidades	Descripción
Interfaz gráfica	Con el objetivo de facilitar la comprensión del juego para el usuario, se tomará la decisión de implementar una interfaz gráfica que le brinde una representación visual del mundo del juego, mejorando así su experiencia y accesibilidad.
Diseño del mundo	Con el propósito de potenciar la inmersión del jugador, se desarrollará un entorno virtual que simula un mundo similar al de Europa, y habrá puntos que serán pueblos a conquistar
Jugabilidad	El juego se concibe como estratégico, siendo responsabilidad del jugador administrar con prudencia sus recursos para alcanzar la victoria. Además, se le ofrece la opción de renunciar a la ruta más directa (opciones de juego) por obtener la máxima puntuación.

Algoritmos a usar	Los algoritmos deben mostrar algo con el fin de aportar al jugador de alguna manera, ya sea que busque aumentar sus puntos, ganar rápidamente o cualquier otra opción.Como algoritmos a utilizar se debe implementar uno que te de la ruta de conquista más rápida y las conexiones entre pueblos más economicas
Pruebas de algoritmos	Por la alta complejidad de representar las conexiones entre las ciudades y además la complejidad de los algoritmos que le aportan algo al jugador, estos se deben probar.
Optimización de rendimiento	El sistema debe ejecutarse en tiempo real con el fin de que la jugabilidad se sienta lo más ágil posible

Se necesita que el software cumpla con las siguientes características:

- Ser un videojuego
- Que funcione mediante una implementación propia de grafos
- El grafo debe estar implementado de dos formas: a modo de lista enlazada o a modo de matriz de adyacencia
- El software debe usar al menos dos algoritmos de grafos, pero tener 6 funcionales (DFS, BSF, Dijkstra, Kruskal, Floyd Warshall y Prim).

Paso 2. Recopilación de Información

Se emplea el mapa de Europa como escenario, con la inclusión de al menos 50 pueblos distribuidos en diversas ubicaciones, representando los lugares que los romanos hasta los vikingos deben conquistar en el juego. Con el objetivo de evitar posibles problemas de censura, se procederá a desvincular los nombres de cada pueblo en el juego.

Grafo:

Un grafo es una estructura de datos que consiste en un grupo de nodos o vértices y aristas, los primeros representando las unidades de los elementos que se buscan representar y las aristas las conexiones de estos elementos. Un ejemplo de un grafo sería una red social, las personas serían los vértices y las relaciones de amistad son aristas que juntan los vértices.

Hay varios tipos de grafos pero los importantes más importantes son los grafos dirigidos y los no dirigidos: en el primero las relaciones no son binarias, es decir, que el vértice “A” tenga una relación “B” con el vértice “C” no es lo mismo que el vértice “C” tenga una relación “B” con el vértice “A”; por el contrario, en el grafo no dirigido, si hay una relación “B” de “A” a “C”, siempre es verdad que hay una relación “B” de “C” a “A”.

Otro tipo de grafo son los grafos con pesos o ponderados, estos indican cierta “resistencia” de llegar de un vértice a otro a través de una arista. Un ejemplo de grafos con pesos serían las carreteras: para llegar de una ciudad a otra hay un coste, un peso, ya sea de tiempo o de consumo de gasolina.

Para representar un grafo se tienen dos formas principales: la lista de adyacencia y la matriz de adyacencia. En el primer tipo, para saber a que nodos se puede acceder el nodo número “i” se tiene que llamar a los valores que hay en esa posición. En el segundo tipo se tiene una matriz que contiene los pesos (si es un grafo sin pesos se pone 1) para llegar del nodo de la fila al de las columnas, un ejemplo sería que al ver la posición (i, j) se obtiene un número (el peso) que dice si se puede llegar del nodo i al nodo j.

GeeksforGeeks. (2023). Graph Data Structure and Algorithms. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Recuperado de https://www.javatpoint.com/java-graph

Hay varias operaciones con grafos como el añadir, quitar o modificar vértices y aristas. Adicionalmente hay varios algoritmos para diferentes funciones:

Recorridos:

DFS: En DFS, se comienza desde un nodo de origen y se explora tan profundamente como sea posible a lo largo de cada rama antes de retroceder. Utiliza una pila (o la recursión) para realizar el seguimiento de los nodos que deben ser visitados.

BFS: En BFS, se comienza desde un nodo de origen y se explora todos sus vecinos antes de pasar a los vecinos de estos nodos. Utiliza una cola para realizar el seguimiento de los nodos que deben ser visitados. A menudo se utiliza para encontrar el camino más corto entre dos nodos en un grafo no ponderado.

En ambos casos la complejidad es O(V + E), donde V es el número de nodos y E es el número de aristas en el grafo.

Murillo, J. (2022, 18 julio). DFS vs BFS.Encora.https://www.encora.com/es/blog/dfs-vs-bfs

Caminos mínimos:

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 o no dirigido, ponderado o no ponderado. Funciona incluso cuando hay aristas con pesos negativos, pero no funciona correctamente en presencia de ciclos negativos. El algoritmo utiliza una matriz para realizar un seguimiento de las distancias mínimas entre todos los pares de nodos. La complejidad del algoritmo es de O(V^3), donde V es el número de nodos en el grafo.

Dijkstra: El algoritmo de Dijkstra se utiliza para encontrar el camino más corto desde un nodo de origen a todos los demás nodos en un grafo ponderado con aristas no negativas. Mantiene una lista de nodos no visitados y, en cada paso, elige el nodo con la distancia mínima conocida desde el nodo de origen. Es más eficiente que Floyd-Warshall para grafos dispersos o de tamaño mediano. La complejidad del algoritmo es O((V + E) * log(V)), donde V es el número de nodos y E es el número de aristas en el grafo.

Best routes selection using Dijkstra and Floyd-Warshall algorithm. (2017, 1 octubre). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/8265662

Subgrafos de mínimo peso:

Prim: Comienza con un nodo arbitrario y selecciona repetidamente la arista más corta que conecta un nodo en el árbol con un nodo fuera de él. Se puede implementar utilizando una cola de prioridad (heap) para seleccionar eficientemente las aristas más cortas. La complejidad del algoritmo es O(E + V * log(V)), donde E es el número de aristas y V es el número de nodos.

Kruskal: Ordena todas las aristas por peso y las agrega al árbol de expansión mínimo en orden ascendente hasta que todos los nodos estén conectados. Utiliza conjuntos disjuntos para verificar si agregar una arista forma un ciclo en el árbol actual. La complejidad del algoritmo es O(E * log(E)), donde E es el número de aristas.

Árboles de peso mínimo: algoritmos de Prim y Kruskal — matemáticas discretas para ciencia de datos. (s. f.). https://madi.nekomath.com/P5/ArbolPesoMin.html

Paso 3. Búsqueda de Soluciones Creativas

Interfaz gráfica:

Los objetivos de la interfaz gráfica son:

- Facilitar el uso y la comprensión por parte del usuario
- Mostrar gráficamente los algoritmos
- Interactuar con los elementos del juego (las ciudades).
- Mejorar la inmersión al juego.

Estructura del Grafo:

El juego presenta un grafo compuesto por al menos 50 nodos y sus respectivas aristas. Cada nodo representa una ubicación, como ciudades, reinos y pueblos, mientras que las aristas indican las rutas disponibles para el avance del ejército romano. Las conexiones entre ubicaciones están ponderadas con un peso que refleja la dificultad de conquistar ese territorio.

Apariencia del mundo del juego:

El único fin de la apariencia del mundo, al no afectar directamente la jugabilidad, es la apariencia e inmersión del usuario, por lo que se debe referenciar a la geografía de Europa y a la edad media.

Jugabilidad:

El juego se basa en la estrategia y gestión de recursos, los recursos son limitados por lo que el jugador se ve forzado a escoger las mejores decisiones. Se progresa mediante la conquista de territorios hasta llegar hasta los vikingos, ya que inicia en Sicilia, el jugador debe atravesar toda Europa (que está dividida por niveles).

Sistema de puntajes:

El sistema de puntajes se basa en los pueblos conquistados junto a los recursos restantes al terminar el juego, este sistema le da aún más importancia a las decisiones en el juego.

Algoritmos Cruciales:

Dos algoritmos cruciales están disponibles para ayudar al jugador:

- Dijkstra y Floyd Warshall, que encuentran el camino con el menor costo hacia la base vikinga.
- Kruskal y Prim, que muestra los 10 caminos con menos peso del árbol de recubrimiento mínimo. Recordemos que los jugadores deben elegir entre eficiencia y expansión del imperio.

Paso 4. Transición de las Ideas a los Diseños Preliminares

Para la interfaz gráfica se pensó en:

- JavaFX
- Java Swing
- SWT
- AWT.

Para la parte de ayudas al jugador se pensó en dos ayudas principales: Obtener la ruta más rápida hacia los vikingos, para lo que se pensó en Dijkstra o Floyd Warshall, y mostrar los caminos más económicos para lo que se pensó en Prim o Kruskal.

Para la apariencia del mundo del juego se propone poner un mapa para mejorar la inmersión del usuario junto a iconos que te representen como conquistador romano y otro que muestre dónde están los vikingos. Adicionalmente se podría integrar música medieval para aún más inmersión del usuario.

Paso 5. Evaluación y Selección de la Mejor Solución

Para evaluar las distintas soluciones se escogieron varios criterios que van del 1 al 5, siendo 5 la mejor calificación y uno la peor.

Usabilidad: ¿La solución es fácil de usar para los usuarios finales? ¿Es la más pertinente para lo que se busca?

Experiencia necesaria: ¿Qué tanta experiencia requiere la solución para que esta sea rápida y óptima al momento de desarrollar?

Experiencia del equipo: ¿Qué tan familiarizado está el equipo con la solución? ¿se ha trabajado con soluciones similares?

Mantenibilidad: ¿Qué tan fácil es modificar la solución? ¿Qué tan flexible es?

Escalabilidad: ¿La solución se adapta mejor a los posibles cambios futuros?

Interfaz gráfica:

	Usabilidad	Experiencia necesaria	Experiencia del equipo	Mantenibilidad	Escalabilidad	Total
Java fx	4	5	4	5	5	19
Java Swing	5	4	2	4	3	16
SWT	5	3	1	3	3	15
AWT	3	2	1	3	2	13

Como se puede observar, para la interfaz gráfica se usará Java fx debido a que: Tiene buena usabilidad, la experiencia necesaria para usarlo es bastante poca y se vuelve aún menor si se usan herramientas como Scene builder, el equipo tiene mayor experiencia con este, es muy mantenible y además es muy escalable.

Ayudas al jugador - Camino mínimo:

	Usabilidad	Experiencia necesaria	Experiencia del equipo	Mantenibilidad	Escalabilidad	Total
Dijkstra	5	5	5	4	4	23
Floyd Warshall	4	4	4	3	4	19

Se optó por Dijkstra debido a que: se acopla mejor a la situación dado que el jugador siempre inicia por un nodo específico, es más fácil de implementar, el equipo tiene más experiencia con este y que es un algoritmo más sencillo y mantenible.

Ayudas al jugador - Caminos con menor peso:

	Usabilidad	Experiencia necesaria	Experiencia del equipo	Mantenibilidad	Escalabilidad	Total
Kruskal	5	4	5	5	5	24

Prim	4	4	5	5	4	22

Se optó por Kruskal porque lo que se busca principalmente son los caminos más económicos, buscar los caminos mínimos entre pares de vértices sería algo innecesario.

Apariencia del mundo:

	Usabilidad	Experiencia necesaria	Experiencia del equipo	Mantenibilidad	Escalabilidad	Total
Mapa de europa	5	4	4	5	5	23
Iconos representativos	3	4	4	5	5	21
Música temática	5	2	5	5	5	22

Se llegó a la conclusión de que el mapa de Europa es la principal solución para “sumergir” al jugador dentro del mundo del juego. Cabe aclarar que ninguna de estas opciones son excluyentes entre sí, no habría problema en implementarlas todas o parte de ellas.