Método de la ingeniería

# Step 1. Problem identification

I drive

*Descripción*:

Es necesario diseñar un sistema que permita la implementación de un juego que en este caso es una especie de laberinto con enemigos. Este juego permitirá al usuario jugar contra la cpu (enemigos). Los cuales competirán en un tipo de juego de casillas, donde cada participante solo puede moverse hacia sus vértices adyacentes.Teniendo en cuenta que el juego necesita ser elaborado mediante la implementación de grafos, cada cuadrado del mapa donde los jugadores pueden moverse se modelará como si fuera un vértice de un grafo.

La historia es la siguiente: Es 1970, y Dani el Rojo, un experto atracador de bancos, se encuentra realizando un atraco rutinario a la Fábrica Nacional de Moneda y Timbre en España. El atraco va bien; Dani ha conseguido hacerse con 200 millones de pesetas. Lo único que falta es salir del banco y llegar a una de sus guaridas donde estará seguro y completará el atraco. Tú eres el conductor elegido para llevar a Dani a alguna de sus guaridas. Dispones de tu DeLorean y tu habilidad al volante. Debes ser muy rápido..

Estructura del Juego

Basado en el hecho de que este juego necesita ser realizado mediante la implementación de grafos, cada cuadrado del tablero, donde los jugadores pueden caer, se modelará como si fuera un vértice.

Componentes del Grafo:

50 Vértices (ubicaciones): Cada uno de los 50 vértices representa una ubicación única en la ciudad, como intersecciones de calles o puntos de interés diferentes (museo, colegio, etc.).

50 Aristas (conexiones entre ubicaciones): Las 50 aristas representan las calles o caminos que conectan las ubicaciones. Cada arista conecta dos vértices y tiene un peso que representa el tiempo de viaje.

Para evitar que la implementación de la estructura del grafo sea desperdiciada, es decir, donde el peso de cada arista de los vértices sea siempre uno, el juego se modelará con diferentes pesos entre los vértices. De esta manera, proporcionará una mejor jugabilidad y permitirá que los algoritmos que implementemos sean eficientes.

Objetivo del Juego

Debes llegar a una de las guaridas de Dani el Rojo en el mejor tiempo posible para que así escape de la policía y obtengas tu recompensa del 20% de la recaudación del golpe.

Jugabilidad

Puedes moverte por cualquier calle; en un atraco no hay leyes ni semáforos. puede moverse solo hacia sus vértices adyacentes.

Condiciones de Victoria

Llegar a una de las guaridas sin que la policía te capture.

*Problem Definition*

El problema es desarrollar un juego que cuente con grafos de con un **mínimo de 50 vértices** y **50 aristas,** y tal que para su solución sea posible aplicar al menos dos (2) de los algoritmos de grafos en este caso Floyd Warshall y BFS. En este caso es necesario crear un juego el cual debe permitir al jugador moverse a los diferentes vértices, hasta que llegue a la guarida(condición de victoria) o en su defecto si es atrapado por un policía(condición de derrota) acabar el juego.

*Identification of needs and symptoms*.

Representación del mapa de juego con Grafos:

Necesidad de una estructura adecuada para representar el mapa de juego como un grafo.Implementación de vértices para representar las ubicaciones y aristas para modelar las conexiones entre ellas. Es decir, es necesario poder agregar vértices y agregar aristas para poder llevar a cabo una representación gráfica del tablero de juego.

Algoritmos de búsqueda de mínimo peso entre vértices:

Necesidad de algoritmos de búsqueda de mínimo peso en entre vértices para modelar los policías y que intenten llegar a los ladrones de la manera más eficiente.

Algoritmos de Búsqueda en Grafos:

Necesidad de algoritmos de búsqueda en grafos para determinar las rutas y movimientos de los jugadores a lo largo del mapa.

*Requirements*

| Client | Marlon |
| --- | --- |
| User | Player of the game |
| Functional requirements | RF1 - Show map  RF2-Allow Movement of player.  RF3-Calculate shortest path.  RF4-Allow victory |
| Problem context | Es necesario diseñar un sistema que permita la implementación de un juego que en este caso es una especie de laberinto con enemigos. Este juego permitirá al usuario jugar contra la cpu (enemigos). Los cuales competirán en un tipo de juego de casillas, donde cada participante solo puede moverse hacia sus vértices adyacentes.Teniendo en cuenta que el juego necesita ser elaborado mediante la implementación de grafos, cada cuadrado del mapa donde los jugadores pueden moverse se modelará como si fuera un vértice de un grafo. |
| Non-functional requirements | * Quick Response * Interfaz gráfica con javaFx |

| Identifier and Name | *RF1 - Show map* | | | |
| --- | --- | --- | --- | --- |
| Resumen | El programa debe ser capaz de representar el mapa con sus vértices y sus aristas. | | | |
| Inputs | **Name** | **Data type** | | **Valid values condition** |
| Nickname | String | |  |
| Result or Postcondition | Se creará un grafo con 50 vértices. | | | |
| Outputs | **Output Name** | | **Data type** | **Format** |
| msg | | String | A message is displayed confirming that the player has been added. |

| Identifier and Name | RF2- Allow Movement of players. | | | |
| --- | --- | --- | --- | --- |
| Resumen | *El programa permite a un jugador moverse de vértice en vértice dando click en el vértice que adyacente que desee.* | | | |
| Inputs | **Name** | **Data type** | | **Valid values condition** |
| click | button | |  |
| Result or Postcondition | The program stores the player. | | | |
| Outputs | **Output Name** | | **Data type** | **Format** |
|  | |  | . |

| Identifier and Name | RF3-Calculate shortest path. | | | |
| --- | --- | --- | --- | --- |
| Resumen | *El sistema debe permitir que el jugador sea perseguido por los policías e intentar que lo atrapen para tener una condición de derrota y darle jugabilidad al juego, en este caso para esto se usa el algoritmo Floyd Warshall.* | | | |
| Inputs | **Name** | **Data type** | | **Valid values condition** |
|  |  | |  |
| Result or Postcondition |  | | | |
| Outputs | **Output Name** | | **Data type** | **Format** |
| msg | | String | “you lose” |

| Identifier and Name | RF4-Allow victory | | | |
| --- | --- | --- | --- | --- |
| Resumen | *El sistema debe permitir que el jugador que el jugador consiga ganar el juego para esto deberá hacer click en el vértice de victoria si es adyacente a el vértice donde se encuentra.* | | | |
| Inputs | **Name** | **Data type** | | **Valid values condition** |
| click | button | |  |
| Result or Postcondition |  | | | |
| Outputs | **Output Name** | | **Data type** | **Format** |
| msg | | String | you win |

## 

## 

## 

## **Step 2. Gathering Information**

*Information about what is required*

*Definitions*

Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices( V ) and a set of edges( E ). The graph is denoted by G(V, E).

Tomado de:

<https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/#what-is-graph-data-structure>

Tipos de grafos:

Dirigido: es aquel que las aristas tienen un único sentido, Los aristas de un grafo dirigido se representan gráficamente con flechas. Sea V un conjunto finito no vacío, y sea la relación binaria E ⊆ V xV . El par ordenado (V, E) es un grafo dirigido sobre V, o digrafo, donde V es el conjunto de vértices o nodos y E es su conjunto de aristas. Escribimos G = (V, E) para denotar tal digrafo.

tomado de: https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso\_Propedeutico/Matematicas\_Discretas/Capitulo\_4\_Grafos.pdf

No Dirigido: Cuando no importa la dirección de las aristas, la estructura G = (V, E), donde E es ahora un conjunto de pares no ordenados sobre V, es decir el conjunto de aristas representa una relaci´on sim´etrica binaria, donde si Vj y Vk son vértices cualesquiera del conjunto de v´ertices V de un grafo, (Vj , Vk) ∈ E −→ (Vk, Vj ) ∈ E. Decimos que tenemos un grafo no dirigido.

tomado de: https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso\_Propedeutico/Matematicas\_Discretas/Capitulo\_4\_Grafos.pdf

Weighted: un grafo donde cada arista tiene asociado un valor para representar un costo.

Recorridos:

Breadth First Search (BFS) is a graph traversal algorithm that explores all the vertices in a graph at the current depth before moving on to the vertices at the next depth level. It starts at a specified vertex and visits all its neighbors before moving on to the next level of neighbors. BFS is commonly used in algorithms for pathfinding, connected components, and shortest path problems in graphs.

Tomado de:<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

Una búsqueda en profundidad (DFS) es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo. Su funcionamiento consiste en ir expandiendo cada uno de los nodos que va localizando, de forma recurrente (desde el nodo padre hacia el nodo hijo). Cuando ya no quedan más nodos que visitar en dicho camino, regresa al nodo predecesor, de modo que repite el mismo proceso con cada uno de los vecinos del nodo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.

tomado de: https://www.encora.com/es/blog/dfs-vs-bfs

**The Floyd-Warshall algorithm:**is a fundamental algorithm in computer science and graph theory. It is used to find the shortest paths between all pairs of nodes in a weighted graph. This algorithm is highly efficient and can handle graphs with both positive and negative edge weights, making it a versatile tool for solving a wide range of network and connectivity problems..

tomado de: <https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/>

Dijkstra: El algoritmo de Dijkstra, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con respecto a los pesos en cada arista. La idea 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, al resto de vértices que componen el grafo, el algoritmo se detiene.

Tomado de: <http://atlas.uned.es/algoritmos/voraces/dijkstra.html#:~:text=El%20algoritmo%20de%20Dijkstra%2C%20tambi%C3%A9n,con%20pesos%20en%20cada%20arista>.

### 

### **Step 3. Creative Solutions Search**

**Alternative 1: Floyd-Warshall, Graph, BFS, Queue**

Floyd-Warshall:

Calcula las rutas más cortas entre todos los pares de vértices en un grafo ponderado.

Permite a los policías determinar la manera más rápida de alcanzar al ladrón, adaptándose a cambios en la posición del ladrón.

Graph:

Representa el mapa del juego con vértices (ubicaciones) y aristas (caminos).

Facilita la visualización y cálculo de rutas en el juego.

BFS (Breadth-First Search):

Busca rutas y explora el grafo desde un nodo de partida.

Útil para que el jugador modele el mapa y determine rutas accesibles rápidamente.

Queue:

Implementa BFS para explorar el grafo nivel por nivel.

Permite una exploración ordenada y sistemática del grafo.

**Alternative 2: Dijkstra, Graph, DFS, HashMap, Stack, Queue, ArrayList**

Dijkstra:

Encuentra las rutas más cortas desde un nodo de origen en un grafo ponderado.

Útil para que los policías determinen la ruta más corta hacia el ladrón en tiempo real.

Graph:

Representa el mapa del juego con vértices y aristas.

Proporciona una estructura clara y manipulable del mapa de juego.

DFS (Depth-First Search):

Explora el grafo para encontrar posibles rutas de escape.

Permite al jugador explorar exhaustivamente el mapa y encontrar caminos largos o complejos.

HashMap:

Almacena y accede rápidamente a la información de los vértices y aristas.

Mejora el rendimiento de los algoritmos de búsqueda y cálculo de rutas.

Stack:

Almacena el recorrido durante la ejecución del DFS.

Facilita el manejo de retrocesos en la búsqueda de rutas.

Queue:

Implementa BFS para explorar el grafo nivel por nivel.

Útil para encontrar rutas más cortas en ciertas situaciones.

ArrayList:

Almacena listas de adyacencia para los vértices y las rutas calculadas.

Proporciona una estructura flexible y dinámica para manejar datos del grafo.

**Step 4. Transition from Ideas to Preliminary Designs.**

**Alternative 1:**

**Floyd Warshall, Graph,BFS,queue**

Floyd-Warshall:

El algoritmo de Floyd-Warshall se emplea para calcular las rutas más cortas entre todos los pares de vértices en un grafo ponderado. En el contexto del juego, esto permitiría a los policías determinar eficientemente la forma más rápida de alcanzar al ladrón, adaptándose de manera reactiva a los cambios en la posición del ladrón y las nuevas conexiones en el grafo.

Graph:

Se utiliza una estructura de grafo para representar el mapa del juego, donde los vértices representan ubicaciones en la ciudad y las aristas representan las calles o caminos entre ellas. Esta representación proporciona una visualización clara del mapa del juego y facilita el cálculo de rutas.

BFS (Breadth-First Search):

BFS se utiliza para buscar rutas y explorar el grafo desde un nodo de partida. Esto sería útil para el jugador, ya que permite modelar el mapa del juego y determinar rápidamente las rutas accesibles desde su ubicación actual. Además, BFS es eficiente para la exploración de grafos, lo que lo hace útil para la navegación en el juego.

Queue:

Implementa el BFS para explorar el grafo nivel por nivel. Permite explorar el grafo de manera ordenada y sistemática.

**Alternative 2:**

**Dijkstra, Graph, DFS,HashMap,Pila ,Cola , Arraylist**

Dijkstra es un algoritmo utilizado para encontrar las rutas más cortas entre un nodo de origen y todos los demás nodos en un grafo con pesos en las aristas. Esto sería útil para los policías para determinar la ruta más corta desde su ubicación actual hasta la posición del ladrón en tiempo real.

Graph:

para representar el mapa del juego. Los vértices representan ubicaciones en la ciudad y las aristas representan las calles o caminos entre ellas.Proporciona una estructura clara y manipulable del mapa de juego.

DFS (Depth-First Search):

Explorar el grafo para encontrar posibles rutas de escape Permite al jugador explorar el mapa de manera exhaustiva, asegurando que se consideren todas las posibles rutas y facilitando la búsqueda de caminos largos o complejos.

HashMap:

Para almacenar y acceder rápidamente a la información de los vértices y aristas, como los pesos de las aristas y las conexiones entre vértices.

Proporciona acceso eficiente a los datos del grafo, mejorando el rendimiento de los algoritmos de búsqueda y cálculo de rutas, especialmente cuando se necesita acceder a la información de manera no secuencial.

Pila (Stack):

para Almacenar el recorrido del jugador o de los policías durante la ejecución del DFS.Facilita la implementación del DFS y el manejo de retrocesos en la búsqueda de rutas, permitiendo almacenar los nodos visitados

ArrayList:

Almacenar listas de adyacencia para los vértices, así como las rutas calculadas y los caminos posibles.

Algo a tener en cuenta es que la mejor opcion podria ser usar Floyd Warshall porque hay que actualizar la posición de los policías cada que el jugador se mueve una casilla, por lo que si usáramos Dijkstra, tocaría hacer un recorrido n^2 cada vez, en cambio con Floyd tenemos la distancia mínima entre cualquier par de vértices desde el inicio.

Step 5. Evaluation and Selection of the Best Solution

**Criterio 1: Eficiencia del**

**[4] Muy eficiente**

**[3] Eficiente**

**[2] Moderadamente eficiente**

**[1] Poco eficiente**

**Criterio 2: Flexibilidad y Escalabilidad**

**[4] Muy flexible y escalable**

**[3] Flexible y escalable**

**[2] Moderadamente flexible y escalable**

**[1] Poco flexible y escalable**

**Criterio 3: Facilidad de Implementación**

**[4] Muy fácil de implementar**

**[3] Fácil de implementar**

**[2] Moderadamente fácil de implementar**

**[1] Difícil de implementar**

**Criterio 4: Exactitud de la Solución**

**[2] Se adapta perfecto a la solución**

**[1] Solución aproximada**

*Evaluation*

|  | Criteria 1 | Criterio 2 | Criterio 3 | Criterio 4 | Total |
| --- | --- | --- | --- | --- | --- |
| Alternativa 1 | 3 | 3 | 3 | 2 | 11 |
| Alternativa2 | 3 | 3 | 2 | 1 | 9 |

*Selection*

In conclusion,alternative 1 is the best option.