Proyecto Algoritmos y estructuras de datos

Betancourt Ramirez Ernesto (A00049172)

**Introducción:**

Se va a realizar un juego cuyo objetivo recorrer y completar todos los niveles gastando el menor número de monedas posibles. Para moverse por el mapa hacia las pantallas hay un costo de monedas que varía según el camino que quiera tomar el jugador. Cada pantalla va a ser un mini juego de trivia en donde se le preguntara algo.

**Definición del problema:**

Generar un mapa basado en un grafo en donde el jugador con una cantidad de monedas pueda moverse por los diferentes caminos de este y llegar a los diferentes nodos los cuales contienen un mini juego para que el jugador gane más monedas y así pueda seguir con otros nodos, el jugador ganará cuando complete todos los nodos(pantallas) y supere la puntuación del mapa.

**Requerimientos:**

|  |  |
| --- | --- |
| Requerimiento funcional 1 | |
| Nombre: | Calcular niveles |
| Descripción: | Se calcularán los niveles del grafo desde donde haya iniciado el jugador |
| Entrada: | Índice del nodo |
| Salida: | Una lista de los niveles de todos los nodos |

|  |  |
| --- | --- |
| Requerimiento funcional 2 | |
| Nombre: | Asignar niveles |
| Descripción: | Asigna un juego con la dificultad asociada al nodo |
| Entrada: | Una lista de los niveles de los nodos |
| Salida: |  |

|  |  |
| --- | --- |
| Requerimiento funcional 3 | |
| Nombre: | Puntuación a superar |
| Descripción: | Asigna una puntuación que el jugador deberá superar para poder ganar |
| Entrada: | Matriz de pesos |
| Salida: | Puntuación a superar |

|  |  |
| --- | --- |
| Requerimiento funcional 4 | |
| Nombre: | Mover jugador |
| Descripción: | Permite mover al jugador de un nodo a otro |
| Entrada: | Posición actual del jugador, posición final |
| Salida: |  |

|  |  |
| --- | --- |
| Requerimiento funcional 5 | |
| Nombre: | Sumar/restar monedas |
| Descripción: | Suma o resta monedas al jugador dependiendo del caso |
| Entrada: | Monedas del jugador, monedas a restar |
| Salida: |  |

|  |  |
| --- | --- |
| Requerimiento funcional 6 | |
| Nombre: | Quitar vidas |
| Descripción: | Quita una vida al jugador si este responde mal una pregunta |
| Entrada: | Vidas del jugador |
| Salida: |  |

|  |  |
| --- | --- |
| Requerimiento funcional 7 | |
| Nombre: | Cargar preguntas y respuestas |
| Descripción: | Asigna una pregunta y una respuesta a cada pantalla, dependiendo del nivel de este |
| Entrada: | Nivel de la pantalla |
| Salida: |  |

|  |  |
| --- | --- |
| Requerimiento funcional 8 | |
| Nombre: | Generar matrices |
| Descripción: | Genera la matriz de adyacencia, pesos y de visitas de las aristas del grafo con valores aleatorios. |
| Entrada: |  |
| Salida: |  |

**Recopilación de información necesaria:**

* Los niveles por lo general se presentan como 2D continuo o espacio 3D, pero en juegos como *Super Mario Bros. 3*, los espacios pueden estar separados, con alguna forma de tele transportación entre ellos. (Fundación Wikimedia, Inc., 2018)
* Un área se usa para definir un nivel que, literalmente, coexiste físicamente entre múltiples niveles, en el que el jugador puede progresar de un “nivel” a otro simplemente usando la física del juego. A pesar de que coexisten espacialmente, cada área presenta sus propios temas, recompensas y desafíos. (Fundación Wikimedia, Inc., 2018)
* El jugador se desplaza durante el juego a través de dos pantallas: un mapa que muestra el mundo y sus respectivos niveles, y el campo de juego del nivel. El mapa del mundo muestra una representación aérea y tiene varios caminos que llevan de la entrada del mundo a un castillo. Las rutas conectan los niveles, fortalezas y otros íconos del mapa y permitirá a los jugadores a tomar rutas diferentes para llegar a la meta de cada mundo. (Fundación Wikimedia, Inc., 2018)
* En matemáticas y ciencias de la computación, un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto. (Fundación Wikimedia, Inc., 2018)
* En Ciencias de la Computación, Búsqueda en anchura (en inglés *BFS* - *Breadth First Search*) es un algoritmo de búsqueda no informada utilizado para recorrer o buscar elementos en un grafo (usado frecuentemente sobre árboles). Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo. A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol. (Fundación Wikimedia, Inc., 2018)
* El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. (Fundación Wikimedia, Inc., 2018)
* En informática, el algoritmo de Floyd-Warshall, descrito en 1959 por Bernard Roy, es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. (Fundación Wikimedia, Inc., 2018)
* Una Búsqueda en profundidad (en inglés DFS o *Depth First Search*) es un algoritmo de búsqueda no informada utilizado para recorrer todos los nodos de un grafo o árbol (teoría de grafos) de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa ([Backtracking](https://es.wikipedia.org/wiki/Backtracking" \o "Backtracking)), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado. (Fundación Wikimedia, Inc., 2018)
* El algoritmo de Prim es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) perteneciente a la [teoría de los grafos](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos) para encontrar un [árbol recubridor mínimo](https://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo) en un [grafo](https://es.wikipedia.org/wiki/Grafo) [conexo](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos#Grafos_conexos), no dirigido y cuyas [aristas](https://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)) están etiquetadas. En otras palabras, el [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) encuentra un subconjunto de [aristas](https://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)) que forman un [árbol](https://es.wikipedia.org/wiki/%C3%81rbol_(teor%C3%ADa_de_grafos)) con todos los [vértices](https://es.wikipedia.org/wiki/V%C3%A9rtice_(Teor%C3%ADa_de_grafos)), donde el peso total de todas las [aristas](https://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)) en el árbol es el mínimo posible. Si el grafo no es conexo, entonces el algoritmo encontrará el [árbol recubridor mínimo](https://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo) para uno de los componentes conexos que forman dicho grafo no conexo. (Fundación Wikimedia, Inc., 2018)
* El algoritmo de Kruskal es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo" \o "Algoritmo) de la [teoría de grafos](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos" \o "Teoría de grafos) para encontrar un [árbol recubridor mínimo](https://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo" \o "Árbol recubridor mínimo) en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un *árbol expandido mínimo* para cada [componente conexa](https://es.wikipedia.org/w/index.php?title=Componente_conexa_(teor%C3%ADa_de_los_grafos)&action=edit&redlink=1" \o "Componente conexa (teoría de los grafos) (aún no redactado))). (Fundación Wikimedia, Inc., 2018)

**Búsqueda de soluciones creativas:**

Para la búsqueda de soluciones creativas me base en una lluvia de ideas para analizar y proponer todo lo posible y dar solución a los diferentes aspectos que tiene el problema.

Una vez propuestas las ideas se las analice para ver si eran buenas, posibles y las agregaba a una lista que sería analizada más a fondo después.

La lista de ideas que salieron fue la siguiente:

Para el mapa:

* Se propuso representar el mapa como un grafo ponderado no direccionado, en donde cada vértice fuera una pantalla y cada arista el camino con un costo para ir de una pantalla a otra.
* Se propuso usar el algoritmo de Dijkstra para mostrarle al jugador los caminos con menor peso, de donde se encuentra el jugador a todos los demás.
* Se propuso usar el algoritmo de Floyd-Warshall para mostrarle al jugador los caminos con menor peso entre todo par de vértices.

Para las pantallas:

* Se propuso que cada pantalla pudiera tener entre tres distintos mini juegos, estos mini juegos serian proyectos o laboratorios pasados de los otros cursos de algoritmos.
* Se propuso que todas las pantallas fueran un mini juego de trivia en donde se le hicieran preguntas al jugador, cada pantalla con una sola pregunta, dependiendo del nivel de las pantallas las preguntas cambiarían de dificultad.
* Se propuso usar el algoritmo BFS para poner los niveles de cada pantalla, ya que con el podemos recorrer el grafo y generar un árbol con este, además de obtener el nivel de cada nodo en el árbol, este nivel seria el nivel de dificultad de cada pantalla, o sea que entre más lejos estén los vértices de la raíz, más difícil será la pantalla.
* Se propuso usar una cola de prioridad para guardar las pantallas dependiendo de su dificultad.

Para el juego:

* Se propuso usar el algoritmo de Prim para generar un puntaje el cual el jugador deberá superar, este puntaje se obtiene dividiendo un número por el peso mínimo de recorrer todo el grafo obtenido con el algoritmo de Prim.
* Se propuso usar el algoritmo de Floyd-Warshall para generar un puntaje el cual el jugador deberá superar.

**Transición de la formulación de ideas a los diseños preliminares:**

Las siguientes propuestas fueron las que se descartaron por distintas razones, principalmente por complicar la solución o no moldearse a lo que se busca:

Para el mapa:

* Usar el algoritmo de Dijkstra para mostrarle al jugador los caminos con menor peso, de donde se encuentra el jugador a todos los demás: esta idea se descartó ya que se buscaba que fuera implementada como un “Buffo” o un “premio” que el jugador pudiese usar a su favor, el problema principal es que el puntaje a superar se obtiene con Prim, y usar Dijkstra no aseguraba que el jugador pudiese obtener una ventaja o beneficio al activarlo.
* Se propuso usar el algoritmo de Floyd-Warshall para mostrarle al jugador los caminos con menor peso entre todo par de vértices: igual que con el de Dijkstra se buscaba usar Floyd-Warshall como un “buffo” u premio, pero si con Dijkstra no aseguraba un beneficio al jugador, con Floyd-Warshall tampoco se aseguraba nada.

Para las pantallas:

* Se propuso que cada pantalla pudiera tener entre tres distintos mini juegos, estos mini juegos serian proyectos o laboratorios pasados de los otros cursos de algoritmos: a pesar de ser una mejor idea frente a las preguntas, era más difícil de implementar, ya que había que adaptar los laboratorios y/o proyectos anteriores a el juego actual, en pocas palabras, se eligió la salida más fácil y menos complicada.
* Se propuso usar una cola de prioridad para guardar las pantallas dependiendo de su dificultad: esta propuesta se descartó ya que no tenía sentido implementarla, o no se supo para que usarla.

Para el juego:

* Se propuso usar el algoritmo de Floyd-Warshall para generar un puntaje el cual el jugador deberá superar: se descartó usar Floyd-Warshall ya que la idea principal fue de usar Prim, ya que este nos da un solo número y no una matriz de ellos.

Las ideas restantes fueron las elegidas para el desarrollo del juego, ya de una u otra manera, permitirán el debido funcionamiento de este, además de que facilitan la implementación del programa.

TAD del grafo:

|  |
| --- |
| TAD Map |
| Lista de adyacencia   |  |  | | --- | --- | | Vértices | C:\Users\Usuario\Desktop\ETIQUETADO3.pngAdyacentes | | a | b, c, d | | b | a, c | | c | a, b | | d | a |   Matriz de adyacencia   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | a | b | c | d | | a | 0 | 1 | 1 | 1 | | b | 1 | 0 | 1 | 0 | | c | 1 | 1 | 0 | 0 | | d | 1 | 0 | 0 | 0 | |
| inv:   * G es un grafo G = (V, E) donde V es un conjunto de vértices o nodos y E un conjunto de las aristas o arcos que relacionan estos nodos. * V ≠ 0 * E es un conjunto de pares no ordenados de elementos de V. |
| Operaciones Primitivas:  getGame: Game X Map: Game setGame: Game X Map:  getPosition: Map X Map: Integer  getSize: Map X Map: Integer  getNodesPositions: Map X Map: lista de nodos |
| |  | | --- | | getGame()  "retorna el juego que el jugador tiene que superar"  {pre: los juegos están cargados} {post: se tendrán las monedas que haya conseguido} | | setGame(entero}  "asigna un juego al nodo"  {pre: hay juegos predefinidos para selección} {post: el nodo tendrá un juego asignado} | | getPosition()  “devuelve la position del nodo actual”  {pre: el jugador ha terminado la partida}  {pos: da la posición} | | getNodesPositions:  “Devuelve una lista con los nodos creados”  {pre: hay un grafo creado}  {pos: se ha retornado la lista} |   **Evaluación y Selección de la Mejor Solución**  Como las propuestas no viables ya fueron descartadas por diferentes motivos, quedaron unas pocas que son las necesarias para el desarrollo del juego, por lo tanto al ser cada una únicas en lo que desarrollan, este paso se va a saltar, ya que no hay necesidad de ver cuál es mejor que otra según un criterio.  Las siguientes son las que sobraron y las que se van a usar:  Para el mapa:   * Se propuso representar el mapa como un grafo ponderado no direccionado, en donde cada vértice fuera una pantalla y cada arista el camino con un costo para ir de una pantalla a otra.   Para las pantallas:   * Se propuso que todas las pantallas fueran un mini juego de trivia en donde se le hicieran preguntas al jugador, cada pantalla con una sola pregunta, dependiendo del nivel de las pantallas las preguntas cambiarían de dificultad. * Se propuso usar el algoritmo BFS para poner los niveles de cada pantalla, ya que con el podemos recorrer el grafo y generar un árbol con este, además de obtener el nivel de cada nodo en el árbol, este nivel seria el nivel de dificultad de cada pantalla, o sea que entre más lejos estén los vértices de la raíz, más difícil será la pantalla.   Para el juego:   * Se propuso usar el algoritmo de Prim para generar un puntaje el cual el jugador deberá superar, este puntaje se obtiene dividiendo un número por el peso mínimo de recorrer todo el grafo obtenido con el algoritmo de Prim. |

<https://es.wikipedia.org/wiki/Nivel_(videojuegos)>

<https://es.wikipedia.org/wiki/Super_Mario_Bros._3>

<https://es.wikipedia.org/wiki/Grafo>

<https://es.wikipedia.org/wiki/Búsqueda_en_anchura>

<https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra>

<https://es.wikipedia.org/wiki/Algoritmo_de_Floyd-Warshall>