Proyecto II
Análisis de Algoritmos y Estructuras de Datos
Grafo de Recursos

Escuela de Ciencias de la Computación e Informática
Facultad de Ingeniería

Autor:

Enrique Ramírez Céspedes

Docente:

MSc. Jonathan Esquivel

03 de diciembre

II Ciclo 2025

1. Introducción

El proyecto consiste en un videojuego en el cual un astronauta debe obtener recursos para escapar del planeta en que se encuentra varado. Para obtener dichos recursos, debe recorrer las aristas de un grafo no dirigido hasta llegar a ciertos objetivos, de tal manera que no se acabe la batería de su traje.

El juego fue programado en `C++`. Para su interfaz gráfica, se usó Raylib.

2. Resultados

El videojuego dispone de una pantalla de bienvenida. Al darle click a **START**, se pasa al ciclo de juego principal. Dentro de este, se muestra el mapa de juego, así como también información relevante para el jugador (nivel de batería, cantidad de recursos, etc.).

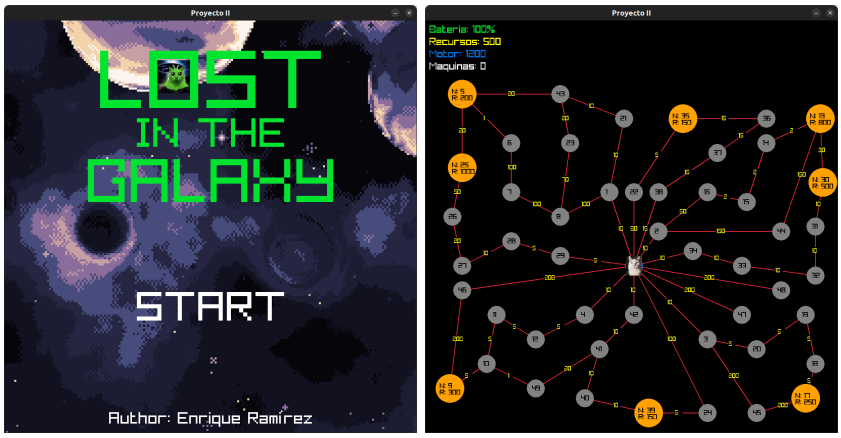

Figura 1: Bienvenida y estado inicial del videojuego

Dentro del ciclo de juego, es posible pausar la partida. Además, se muestra una pantalla de derrota si el nivel de la batería llega a cero. Si la cantidad de recursos es suficiente para armar la nave, se gana la partida y se muestra una pantalla de victoria.

3. Implementación

Se decidió trabajar con Raylib en C++ ya que esa fue la dinámica bajo la cual se desarrolló el Proyecto Integrador de Lenguaje Ensamblador del semestre en curso. Asimismo, esto permitió tener una noción más clara de qué clases implementar.

En síntesis, se tiene la clase `main`. Dentro de esta, se gestiona la ventana de juego y el ciclo de juego (mostrar pantallas de bienvenida, derrota, victoria, etc.). Además, gestiona la lectura de archivos.

Cabe resaltar que se replanteó el sistema de archivos tal que existan dos métodos que lo gestionen: uno lee un archivo de nodos y otro un archivo de aristas. El formato de ambos archivos es `.csv`. Se hizo este cambio puesto que se consideró que era más intuitivo, además de sencillo de implementar.

La clase `Game`, gestiona la inicialización, renderizado y actualización de `Graph`, `Player` y `Pipe`. `Graph` es la clase que representa al mapa de juego.

4. Respuestas a las Preguntas Planteadas

A continuación, se responden las preguntas planteadas en el enunciado del proyecto.

4.1. Análisis de Optimalidad

4.1.1. Sobre Máquina BFS/DFS

1. **¿Por qué este algoritmo casi siempre genera la peor ganancia?**

   Esto sucede porque son algoritmos que no toman en cuenta el peso de las aristas para formar el recorrido, de tal manera que, si bien el costo de implementar una de estas máquinas es el más accesible, el beneficio

obtenido de ellas puede resultar poco significativo si el recorrido que toman para trasladar los recursos a la base tiene nodos con pesos elevados.

2. **¿Garantiza BFS o DFS encontrar algún camino? Sí/No y por qué**

   Sí lo garantizan. Probablemente no sea el mejor, pero lo hacen, ya que no terminan su ciclo hasta llegar al objetivo.

3. **Explique cómo la ignorancia de los pesos lleva a rutas costosas**

   Como no considera los pesos, puede que termine haciendo el recorrido por donde hay una arista muy pesada en lugar de por otra ruta con aristas baratas.

4. **Muestre una captura de pantalla de una tubería nivel 1, que tome un camino visiblemente absurdo y costoso en su mapa**

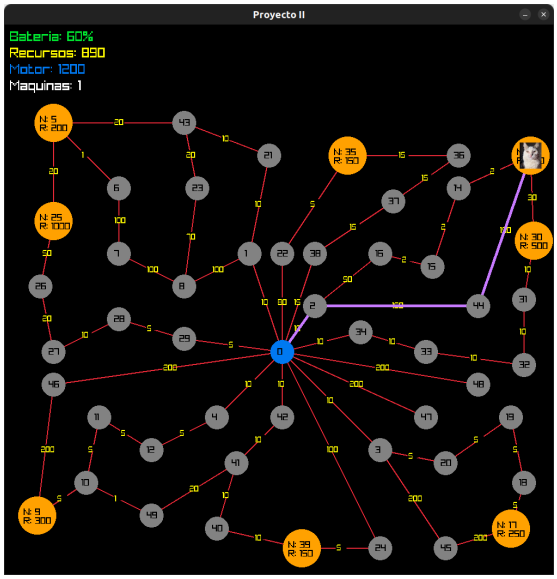

Figura 2: Toma un recorrido de 150 + 150 en lugar de uno (bastante) más liviano

4.1.2. Sobre Máquina Greedy

1. **¿Por qué este algoritmo es mejor que el BFS/DFS, pero aun así no es perfecto?**

   Es mejor porque sí toma en cuenta los pesos. Sin embargo, no es perfecto, pues solo toma en consideración el peso de las aristas conectadas al nodo en que esté procesando el algoritmo durante una determinada iteración. Entonces, puede que la ruta comience siendo muy buena, pero que en algún momento nos lleve al caso donde las aristas tengan mucho peso.

2. **Describa y compare los conceptos de óptimo local y óptimo global**

   El óptimo local es la mejor decisión que puede tomar el algoritmo durante la iteración en que se encuentra. El global es el mejor camino posible en general.

3. **Diseñe o encuentre un escenario en su mapa donde el algoritmo greedy tome una mala decisión**

   Ir del nodo 35 al origen:

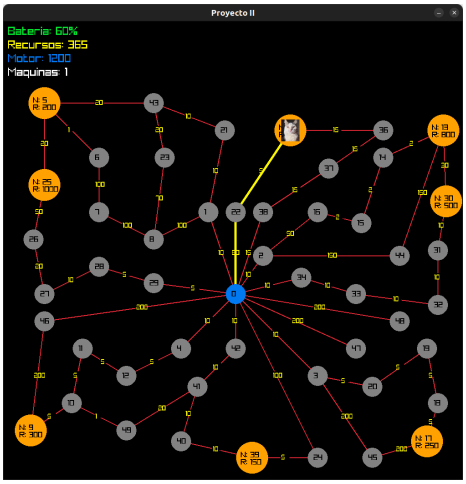

Figura 3: Recorrido Greedy ineficiente

4.1.3. Sobre Máquina Dijkstra

1. **¿Por qué este algoritmo siempre encuentra el camino más rentable?**

   Porque explora todas las rutas posibles y escoge la de menor peso.

2. **Explique cómo explora el grafo en comparación con los otros dos.**

   No ignora pesos ni se ve limitado por óptimos locales.

4.2. Análisis de Complejidad Temporal

1. **Declare la complejidad temporal (Big O) de BFS, DFS y Dijkstra en función de los vértices y los aristas.**

   - O(V+E) para las dos primeras
   - $O(V^2)$ para Dijkstra

2. **Si Dijkstra es más lento de calcular, ¿por qué se utiliza?**

   Porque da la mejor ruta.

4.3. Análisis de Estructuras de Datos

1. **¿Cómo influye la densidad del mapa (cantidad de aristas) en la elección entre una matriz y una lista de adyacencia?**

   En un grafo con pocas aristas, es mejor usar la lista porque usa menos memoria.

2. **¿Cómo habría cambiado el rendimiento (memoria/velocidad) de los algoritmos si hubieran elegido la otra estructura?**

   Hubiera sido más lento, pero en retrospectiva, creo que hubiera sido más fácil para mí programarlo.

5. Conclusiones

Se pudo implementar lo solicitado satisfactoriamente. El proceso resultó útil para profundizar en los temas vistos en el curso.

6. Notas

Se puede acceder al repositorio de control de versiones del proyecto a través del siguiente enlace: `https://github.com/enriqueramirezc/resource-graph`

Referencias

[1] Raylib. (2025). Raylib CheatSheet. https://www.raylib.com/cheatsheet/cheatsheet.html

[2] Enrique Ramírez et. al. (2025). Pixel Pong. https://github.com/Ryusui777/Pixel_Pong_PI_Ensamblador_Grupo04