# Generador y Resolución de Laberintos con Algoritmos de Búsqueda (DFS/BFS)

**Descripción General:**

Este proyecto tiene como objetivo desarrollar un sistema para **generar laberintos aleatorios** y resolverlos utilizando **algoritmos de búsqueda**, concretamente **Búsqueda en Profundidad (DFS)** y **Búsqueda en Anchura (BFS)**. El enfoque principal será tanto la creación del laberinto como la implementación de algoritmos para encontrar el camino desde una posición inicial hasta una meta. Además, se implementará una visualización gráfica del proceso de generación y resolución del laberinto, permitiendo a los estudiantes observar cómo funcionan los algoritmos en tiempo real. Una extensión opcional consistirá en la implementación y comparación de otros algoritmos, como **A**\* o **Dijkstra**, para analizar el rendimiento y la eficiencia en distintos tipos de laberintos.

**Objetivos del Proyecto:**

1. **Generación de laberintos aleatorios** utilizando técnicas apropiadas para crear laberintos con caminos y obstáculos bien definidos.
2. **Implementación de algoritmos de búsqueda (DFS/BFS)** para resolver el laberinto, buscando un camino desde la entrada hasta la salida.
3. **Visualización gráfica** del laberinto y del proceso de resolución, mostrando cómo los algoritmos exploran el espacio de búsqueda.
4. **Comparación de rendimiento** entre diferentes algoritmos de búsqueda, como DFS, BFS, A\* y Dijkstra, en términos de eficiencia (tiempo y número de pasos).
5. **Estructuración del proyecto utilizando Programación Orientada a Objetos (POO)**, con clases que representen el laberinto, los nodos y los algoritmos de resolución.

**Instrucciones Detalladas:**

**1. Generación del Laberinto Aleatorio**

* Se debe diseñar una **clase Laberinto** que se encargue de generar el laberinto aleatoriamente:
  + **Dimensiones del laberinto**: El laberinto será una matriz (por ejemplo, 20x20) donde cada celda puede ser un camino libre o un obstáculo (pared).
  + **Generación del laberinto**: Utilizar algoritmos clásicos para generar laberintos, como el algoritmo de "backtracking" o el método de **Prim** o **Kruskal**. Estos métodos garantizan que haya un camino desde el punto de entrada hasta el punto de salida.
  + **Representación del laberinto**: El laberinto se puede representar como una matriz bidimensional donde:
    - 0 representa un camino libre.
    - 1 representa una pared.
    - Otros valores pueden representar la entrada, la salida o el recorrido del algoritmo.

**2. Visualización del Laberinto**

* Crear una **interfaz gráfica** para representar el laberinto y el proceso de resolución:
  + **Usar Pygame** u otra librería gráfica para dibujar el laberinto en la pantalla.
  + Cada celda del laberinto se dibuja como un rectángulo de color (p. ej., blanco para caminos libres, negro para paredes).
  + Dibujar la posición de inicio (p. ej., un cuadrado verde) y la meta (p. ej., un cuadrado rojo).
  + Permitir que el laberinto se visualice antes y después de ser resuelto, con los pasos del algoritmo mostrados en tiempo real.

**3. Implementación de Algoritmos de Búsqueda (DFS/BFS)**

* Crear una **clase Algoritmo** que implemente los algoritmos de búsqueda:
  + **DFS (Depth-First Search)**: Implementar la búsqueda en profundidad, donde el algoritmo explora tan lejos como sea posible a lo largo de un camino antes de retroceder y probar otras opciones.
    - Utilizar una **pila** para gestionar el proceso de búsqueda.
    - Resaltar el proceso de exploración en la visualización gráfica.
  + **BFS (Breadth-First Search)**: Implementar la búsqueda en anchura, donde el algoritmo explora todos los nodos en el nivel actual antes de pasar al siguiente.
    - Utilizar una **cola** para gestionar la exploración.
    - Comparar el rendimiento de BFS frente a DFS, especialmente en términos de número de pasos y eficiencia.
  + **Mostrar en la visualización** cómo el algoritmo explora el laberinto en tiempo real, cambiando el color de las celdas visitadas durante la búsqueda.

**4. Comparación de Algoritmos de Resolución**

* Para hacer el proyecto más interesante, se debe permitir a los estudiantes **comparar el rendimiento** de los distintos algoritmos:
  + **DFS vs BFS**: Comparar en términos de:
    - Número de pasos dados hasta encontrar la solución.
    - Tiempo de ejecución.
  + **Extensión opcional**: Implementar el **algoritmo A**\*, que utiliza una heurística para encontrar la ruta más corta con mayor eficiencia, o el **algoritmo de Dijkstra**, que siempre encuentra el camino más corto pero sin heurística.
  + Mostrar métricas de rendimiento (como tiempo o pasos) en pantalla para cada algoritmo.

***5. Extensión Opcional: Algoritmos Adicionales (A, Dijkstra)*\***

* **A**\*:
  + A\* es un algoritmo que combina los enfoques de BFS y Dijkstra. Utiliza una **función heurística** para estimar la distancia desde el nodo actual hasta el nodo meta, mejorando la eficiencia en comparación con DFS y BFS.
  + Implementar una función heurística simple, como la **distancia Manhattan**.
* **Dijkstra**:
  + Implementar el **algoritmo de Dijkstra**, que busca el camino más corto desde el nodo de inicio hasta la meta basándose en el peso de las aristas (en este caso, la distancia).
  + Comparar este algoritmo con A\* en términos de eficiencia y complejidad.

**6. Interacción del Usuario**

* Permitir que el usuario seleccione las dimensiones del laberinto y el algoritmo que desea utilizar para resolverlo.
* Añadir un botón o una tecla para generar un nuevo laberinto aleatorio.
* Permitir pausar/reanudar el proceso de búsqueda para que el usuario pueda observar cada paso del algoritmo.

**Requisitos del Proyecto:**

1. **Generador de laberintos** funcional que cree laberintos aleatorios utilizando algún algoritmo apropiado (backtracking, Prim, Kruskal).
2. Implementación de **algoritmos DFS y BFS** para la resolución de laberintos.
3. **Visualización gráfica** del laberinto y del proceso de búsqueda en tiempo real.
4. Extensión opcional: Implementación de **algoritmos A**\* y/o **Dijkstra** para la resolución óptima y comparación de su rendimiento con los algoritmos básicos.
5. **Interfaz interactiva** donde el usuario pueda generar nuevos laberintos y seleccionar el algoritmo de resolución.

**Evaluación del Proyecto:**

* **Generación correcta del laberinto** (diseño, aleatoriedad, caminos conectados) – 30%
* **Implementación de algoritmos de búsqueda (DFS, BFS)** – 30%
* **Calidad de la visualización gráfica** (interfaz clara, pasos visibles) – 20%
* *Extensión opcional (A, Dijkstra)*\* y análisis comparativo – 10%
* **Interacción y experiencia del usuario** – 10%