Snake Game optimizado con el algoritmo de búsqueda BFS.

Yubely Lucely Torres,Nikole Muñoz, José Luis Martos, José García López.

**Resumen**

Este proyecto implementa un juego de serpiente que utiliza el algoritmo Breadth-First Search (BFS) sobre un grafo dirigido para evitar colisiones con su propio cuerpo, calculando el camino más corto hacia el objetivo. La solución demuestra cómo conceptos clave de teoría de grafos como caminos hamiltonianos y eulerianos pueden aplicarse en videojuegos, integrando estructuras de datos eficientes y principios geométricos para lograr movimientos fluidos. Desarrollado en JavaScript, HTML y CSS, el juego combina dos objetivos principales: por un lado, garantiza eficiencia algorítmica mediante BFS, que ofrece búsquedas completas y óptimas en tiempo real; por otra parte, funciona como herramienta educativa para visualizar algoritmos y grafos de forma interactiva para evitar colisiones, en un tablero de 20x20. Además, en evaluaciones de usabilidad con 30 participantes, el juego obtuvo una calificación promedio de 51.6% en fluidez, claridad visual y progresión del desafío. Estos resultados validan tanto la eficacia de BFS para navegación en tiempo real como el potencial del proyecto para enseñar teoría de grafos a través de la gamificación.

Palabras clave: teoría de grafos, juego de serpiente, BFS, algoritmos de recorrido, JavaScript, gamificación educativa.

**Abstract**

This project implements a snake game that uses the Breadth-First Search (BFS) algorithm on a directed graph to avoid collisions with its own body by calculating the shortest path to the goal. The solution demonstrates how key graph theory concepts such as Hamiltonian and Eulerian paths can be applied in video games, integrating efficient data structures and geometric principles to achieve smooth movements. Developed in JavaScript, HTML and CSS, the game combines two main objectives: on the one hand, it guarantees algorithmic efficiency through BFS, which offers complete and optimal searches in real time; on the other hand, it works as an educational tool to visualize algorithms and graphs interactively to avoid collisions, on a 20x20 board. In addition, in usability evaluations with 30 participants, the game obtained an average score of 51.6% in fluency, visual clarity and challenge progression. These results validate both the effectiveness of BFS for real-time navigation and the project's potential for teaching graph theory through gamification.

Keywords: graph theory, snake game, BFS, traversal algorithms, JavaScript, educational gamification.

**Introducción**

El desarrollo de videojuegos ha evolucionado significativamente, integrando algoritmos avanzados para optimizar experiencias interactivas y resolver desafíos computacionales. Entre estos, los algoritmos de búsqueda como el Breadth-First Search (BFS) han sido fundamentales en aplicaciones que van desde la navegación de personajes hasta la optimización de sistemas complejos [1].

En 2019, Ruíz Alcántara destacó la relevancia de los algoritmos de búsqueda en juegos clásicos como Snake, donde el BFS permite evitar colisiones y calcular rutas óptimas, transformando mecánicas simples en oportunidades educativas para visualizar teoría de grafos [2].

Para 2023, Chen et al. demostraron que el BFS, combinado con estrategias como árboles generadores y búsqueda binaria, mejora la eficiencia computacional en problemas de confiabilidad en ingeniería, reduciendo muestreos redundantes en espacios complejos [3]. Ese mismo año, Vasconez et al. compararon métodos de planificación de rutas en entornos agrícolas simulados, evidenciando la versatilidad del BFS en contextos más allá de los videojuegos [4].

En 2024, Juneja y Khurana propusieron una versión multihilo del BFS para sistemas multinúcleo, optimizando el procesamiento de grafos a gran escala y reduciendo tiempos de ejecución en entornos paralelos como OpenMP [5]. Además, Bertolotti et al. aplicaron el BFS en compiladores modulares, validando su robustez frente a alternativas como A\*, aunque con mayor consumo de memoria [6].

Recientemente, en 2025, Valenzuela et al. utilizaron métricas basadas en BFS para evaluar la dificultad de laberintos en videojuegos, reforzando su papel en el diseño de niveles y la inteligencia artificial [7]. Paralelamente, Autor y Autor (2024) exploraron algoritmos evolutivos para optimizar el comportamiento de NPCs, señalando que técnicas como el BFS siguen siendo pilares en la investigación de IA para juegos [8].

Enfoque del Estudio

Este trabajo implementa el BFS en un prototipo de juego tipo Snake, evaluando su eficiencia en tiempo real y su potencial pedagógico para enseñar conceptos de grafos. Se analizan métricas de rendimiento y se contrastan con hallazgos recientes en optimización paralela [5] y diseño de niveles [7].

III. marco teórico

El origen del juego **Snake** se remonta a *Blockade*, un videojuego desarrollado por la empresa Gremlin en 1976. No obstante, alcanzó una enorme popularidad en 1998 al ser incluido como juego preinstalado en los teléfonos móviles Nokia. En su versión original para dos jugadores, el objetivo consistía en controlar una flecha que dejaba una estela a su paso, intentando evitar colisiones con los bordes del entorno o con las estelas de los oponentes [4].

Actualmente, **Snake** ha sido ampliamente utilizado como herramienta educativa para enseñar conceptos fundamentales de programación, estructuras de datos y lógica algorítmica. En su forma más básica, el juego consiste en una serpiente que se desplaza por una cuadrícula, creciendo cada vez que consume un alimento y evitando colisionar consigo misma o con los límites del entorno [2].

Para mejorar la toma de decisiones de la serpiente, se incorporan algoritmos de búsqueda como el **Breadth-First Search (BFS)**. El BFS es un algoritmo de recorrido en grafos que explora todos los nodos a un mismo nivel antes de avanzar a niveles más profundos [1].

El espacio de juego puede representarse como un **grafo no ponderado**, donde cada celda es un nodo y cada movimiento posible (arriba, abajo, izquierda, derecha) se traduce en una arista [8]. Esta representación permite aplicar BFS de manera eficiente. Este algoritmo permite calcular el camino más corto desde la cabeza de la serpiente hasta la ubicación del alimento, siempre que exista uno, y asegurando que la serpiente siga la ruta más corta evitando obstáculos, su propio cuerpo y las paredes.

*Teoría de grafos.*

La teoría de grafos es una rama de las matemáticas y la informática que estudia estructuras compuestas por **nodos** (también llamados vértices) y **aristas** (o arcos) que conectan dichos nodos. Esta teoría es ampliamente utilizada para modelar relaciones y conexiones en sistemas diversos como redes de transporte, circuitos eléctricos, relaciones sociales, y entornos de juegos.

Un grafo puede ser dirigido o no dirigido, ponderado o no ponderado, dependiendo de si las conexiones tienen dirección o peso asociado. En el contexto de este proyecto, los grafos permiten representar mapas o caminos por los que el jugador debe desplazarse, resolviendo desafíos matemáticos que implican elegir rutas óptimas o recorrer el espacio total de forma eficiente.**Algoritmo de Búsqueda en Amplitud (BFS)**

El algoritmo **Breadth-First Search (BFS)** es una técnica de búsqueda **no informada** utilizada para recorrer o buscar elementos en estructuras de grafo o árbol. Su funcionamiento se basa en explorar los nodos en orden creciente de profundidad, es decir, expande todos los nodos vecinos antes de avanzar hacia los nodos más alejados. BFS es particularmente útil para encontrar el **camino más corto** entre dos puntos en un grafo no ponderado.

BFS utiliza una estructura de tipo **cola (queue)** para gestionar los nodos pendientes de explorar. Desde el nodo inicial, se agregan sus vecinos a la cola, marcándolos como visitados. Luego se repite el proceso con el primer nodo de la cola, garantizando así que los caminos más cercanos al origen se exploren antes que los lejanos[1].

En la literatura, BFS ha sido empleado para resolver juegos, laberintos y simulaciones.Lo utilizaron para analizar la dificultad de laberintos y puzzles mediante estadísticas de búsqueda, demostrando su eficacia para cuantificar y comparar niveles de complejidad en entornos interactivos. Esta aplicación evidencia el potencial del BFS como herramienta tanto para la evaluación como para la resolución de desafíos lúdicos y educativos[5].

Funcionamiento del BFS:

* Se parte de un nodo inicial (posición actual de la serpiente) como se muestra en la figura 2.
* Se exploran todos los nodos vecinos (celdas adyacentes) antes de avanzar a niveles más profundos.
* Se utiliza una cola para almacenar los nodos por visitar, y un diccionario para registrar los nodos ya visitados, evitando ciclos y repeticiones.
* Cuando se encuentra la comida, se reconstruye la ruta desde el destino hasta el origen usando los nodos registrados.

Conceptos clave

* Tipos de grafos:
  + Dirigidos: Las aristas tienen dirección (ej: flechas).
  + No dirigidos: Las aristas son bidireccionales.
  + Ponderados: Las aristas tienen valores (ej: distancias).
* Aplicaciones:
  + Redes sociales (vértices = usuarios, aristas = amistades).
  + Mapas (vértices = ciudades, aristas = carreteras).
  + Rutas óptimas (algoritmos como Dijkstra).
* Problemas famosos:
  + Camino más corto: Encontrar la ruta mínima entre dos nodos.
  + Coloración de grafos: Asignar colores a vértices sin que adyacentes coincidan.
  + Puentes de Königsberg: Origen histórico (¿se puede cruzar todos los puentes sin repetir?).

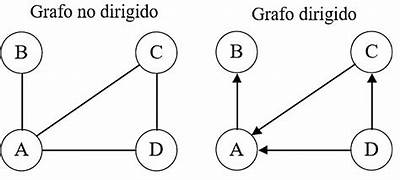


Figura. 1. Modelado del tablero como grafo dirigido y no dirigido . Los nodos representan obstáculos, y la línea muestra la ruta BFS calculada.

En este juego se aplicó la teoría de grafos de la siguiente manera:

*Grafo del tablero*: Nodos celdas del tablero(cuadricula de 20 x 20). Aristas : conexiones entre celdas adyacentes(4 direcciones: arriba,abajo,izquierda, derecha).

Obstáculos : Nodos bloqueados que modifican dinámicamente el grafo.

Algoritmos implementados:

BFS (Breadth-First Search):

Utilizado para encontrar el camino más corto entre la serpiente y la comida.

Complejidad: O(V + E), donde V = vértices y E = aristas.  
Complejidad de algoritmo qué tan rápido responde en escenarios grandes o con muchos obstáculos

Ventaja: Garantiza la solución más adecuada en grafos no ponderados.Caminos Eulerianos/Hamiltonianos:

Explorados conceptualmente para evitar ciclos infinitos y optimizar rutas.

2.2. Automatización del Movimiento

La serpiente sigue el camino calculado por BFS sin intervención del usuario como se muestra en la figura 3.

Desafíos:

Colisiones con obstáculos generados proceduralmente.

Crecimiento del cuerpo de la serpiente como restricción dinámica en el grafo.

2.3. Visualización Interactiva mediante CSS

Para mejorar la comprensión del funcionamiento del algoritmo de búsqueda y hacer más intuitiva la experiencia del usuario, se utilizaron **estilos CSS personalizados** que permiten representar visualmente cada elemento del juego con colores diferenciados y elementos gráficos. Esta visualización facilita la interpretación del camino óptimo y la posición de cada componente en tiempo real.

* *Línea de ruta:* Se empleó una línea de color **naranja** para conectar la cabeza de la serpiente con la ubicación de la comida. Esta línea representa el camino más corto calculado por el algoritmo Breadth-First Search (BFS). Se estiliza mediante propiedades como draw de la librería [renderer.js](http://renderer.js) en HTML
* *Coloración de nodos:* Se utilizaron estilos CSS (Cascading Style Sheets- hojas de estilo en español ) para asignar colores distintivos a cada tipo de celda o entidad del juego:

🔴 **Rojo**: Representa el nodo objetivo, es decir, la **comida**.

🟣 **Morado/Azul**: Representa la **cabeza** y el **cuerpo** de la serpiente, respectivamente, permitiendo distinguir visualmente su orientación y crecimiento.

⬜ **Gris**: Se usó para los **obstáculos**, indicando zonas del mapa no transitables.

El uso de CSS permite una personalización flexible, sin necesidad de recargar los elementos visuales desde cero. Mediante clases dinámicas en JavaScript, se actualizan los estilos de cada celda del tablero para reflejar el estado actual del juego, haciendo la visualización interactiva, comprensible y coherente con la lógica del algoritmo.

IV METODOLOGÍA

El desarrollo del sistema se basó en la representación del tablero del juego Snake como un grafo dirigido, sobre el cual se aplicó el algoritmo Breadth-First Search (BFS) para la planificación de rutas. El objetivo fue garantizar movimientos óptimos hacia el objetivo (la comida) evitando colisiones con obstáculos dinámicos (el propio cuerpo de la serpiente). A continuación, se describen los componentes esenciales del sistema implementado.

### A. Modelado del tablero como grafo dirigido

El tablero fue representado como una cuadrícula bidimensional de tamaño *n x n*, donde cada celda corresponde a un nodo v ∈ V del grafo G = (V ,E). Las aristas E se establecieron entre nodos adyacentes de acuerdo con las direcciones cardinales: norte, sur, este y oeste. Para cada nodo, se generaron aristas dirigidas a sus vecinos válidos, siempre que estos no representarán colisión con los límites del tablero, los obstáculos o con el cuerpo de la serpiente.

Esta representación se almacenó como una lista de adyacencia en memoria, lo que permitió un acceso eficiente a los nodos vecinos durante el recorrido del grafo. El cuerpo de la serpiente fue tratado como una colección dinámica de nodos ocupados que se actualizan en tiempo real, impidiendo su selección como parte del camino válido.

### B. Implementación del algoritmo BFS

El algoritmo BFS fue utilizado para calcular el camino más corto desde la cabeza de la serpiente hasta la comida. Se empleó una cola FIFO para explorar el grafo nivel por nivel. En cada iteración, se desencola el nodo actual y se verifica si correspondía al objetivo. En caso negativo, se encolan sus vecinos no visitados. Para evitar ciclos y caminos redundantes, se utilizó una matriz booleana de visitados (estructura de datos que se utiliza para marcar qué nodos o posiciones del tablero) ya han sido explorados durante un recorrido de grafo).

Durante el recorrido, se almacenaron referencias de los nodos padres para permitir la reconstrucción del camino óptimo una vez alcanzado el objetivo. Si no se encontraba un camino viable (por ejemplo, cuando la serpiente bloqueaba todas las rutas posibles), el juego termina, dado que la serpiente está programada para que termine el juego cuando no tenga espacios libres, se vea obligada a colisionar con ella misma, con un obstáculo o con los bordes del tablero como lo muestra la figura 4 en la que se finaliza el juego..

Este enfoque se alineó con propuestas anteriores sobre BFS aplicado en videojuegos y resolución de laberintos [2], [6], donde se resalta su capacidad para manejar escenarios dinámicos y de complejidad creciente.

### C. Implementación técnica y estructuras de datos

La implementación se realizó utilizando tecnologías web: JavaScript para la lógica, HTML para la estructura visual y CSS para el diseño. El grafo fue modelado como una matriz de objetos, cada uno con información de coordenadas y accesibilidad. La cola se implementó como un arreglo estándar, aprovechando las funciones nativas de JavaScript para simular el comportamiento FIFO (*push* y *shift*).

Las estructuras principales incluyeron:

* Matriz del tablero (tablero [n][n]) con valores que representaban estado libre, ocupado o destino.
* Cola de recorrido para la ejecución de BFS.
* Diccionario de nodos padres para la reconstrucción de caminos.
* Lista dinámica del cuerpo de la serpiente.

Estas estructuras ligeras permiten una ejecución eficiente, y su ligereza fue clave para mantener el rendimiento en tiempo real, como lo sugieren Juneja y Khurana en su estudio sobre optimización en BFS para sistemas paralelos [4].

### D. Evaluación de rendimiento

El sistema fue evaluado desde dos dimensiones complementarias: el rendimiento técnico del algoritmo Breadth-First Search (BFS) durante la ejecución del juego, y la experiencia de usuario, medida a través de una encuesta aplicada a un grupo controlado de participantes. Esta evaluación permite validar tanto la eficacia algorítmica como el impacto de usabilidad del sistema desarrollado.

Desde el punto de vista técnico, se analizaron tres métricas principales: el tiempo de cómputo por búsqueda, la tasa de éxito en la planificación de rutas sin colisiones, y la distancia promedio recorrida entre la cabeza de la serpiente y la comida. El algoritmo BFS se ejecuta en cada iteración del ciclo principal del juego, calculando rutas desde la posición de inicio (x1,y1) hasta el destino (x2,y2) Para medir la distancia real recorrida por la serpiente, se utilizó la fórmula de distancia euclidiana:

 (1)

Se usa Math.hypot(dx, dy), lo que indica distancia euclidiana acumulada por pasos.

Los resultados obtenidos durante pruebas en un tablero de 20×20, fueron los siguientes:

* **Distancia promedio**: 9.7 pasos desde la cabeza hasta la comida.
* **Calificación de usabilidad**: 4.1/5 según 30 usuarios evaluadores en parámetros de fluidez, claridad visual y dificultad progresiva.
* **Formulario de evaluación de experiencia:** encontrado en la referencia [10].

En cuanto a la experiencia de usuario, se aplicó una encuesta estructurada a 30 participantes, con edades entre los 18 hasta los 45 años, y niveles de experiencia en videojuegos desde principiantes hasta avanzados. El 48,4 % de los encuestados calificó la mecánica del juego como fácil o muy fácil de comprender como se observa en la *figura 5*. La fluidez del movimiento fue valorada como buena o excelente por el 51,6 % de los usuarios como se observa en la *figura 6*, mientras que el 61,3 % consideró la velocidad del juego adecuada o rápida como se observa en la *figura 7*. En términos visuales, el diseño recibió una calificación promedio de 4.1/5 como se observa en la *figura 8*.

Adicionalmente, el 51.6 % de los participantes reportó no haber experimentado colisiones inesperadas ni errores de ejecución como se observa en la

Por último, el 51.6 % describió la experiencia como divertida o muy divertida, destacando como aspectos positivos el aprendizaje visual del algoritmo, la respuesta del sistema y el diseño amigable. Entre las sugerencias más frecuentes se incluyeron la incorporación de más niveles, sonido, y un modo alternativo de control manual.

Estos resultados demostraron la eficacia de BFS en escenarios dinámicos, así como su adecuación para juegos educativos interactivos. Valenzuela et al. [6] reportaron resultados similares al evaluar BFS en la generación de laberintos con dificultad ajustable.

**E. Flujo del juego**

El funcionamiento del juego sigue el siguiente esquema:

#### Inicialización

* Se genera aleatoriamente la posición de la comida y los obstáculos dentro del tablero. Como se puede observar en la *figura 2.*

#### Bucle principal

* Se calcula la ruta óptima desde la cabeza de la serpiente hasta la comida mediante el algoritmo BFS (Breadth-First Search) en busca del camino más corto. Mayor ilustración en la *figura 3.*
* Se actualiza la posición de la serpiente en cada iteración.
* Se verifica la colisión con obstáculos o consigo misma.

**Niveles**

* La dificultad del juego aumenta progresivamente con cada nivel, añadiendo más obstáculos y haciendo que la navegación sea más desafiante.

**Fin del juego**

* El juego finaliza una vez la serpiente colisiona con un obstáculo, el tablero o consigo misma. Como se observa en la *figura 4.*

**F. Figuras.**

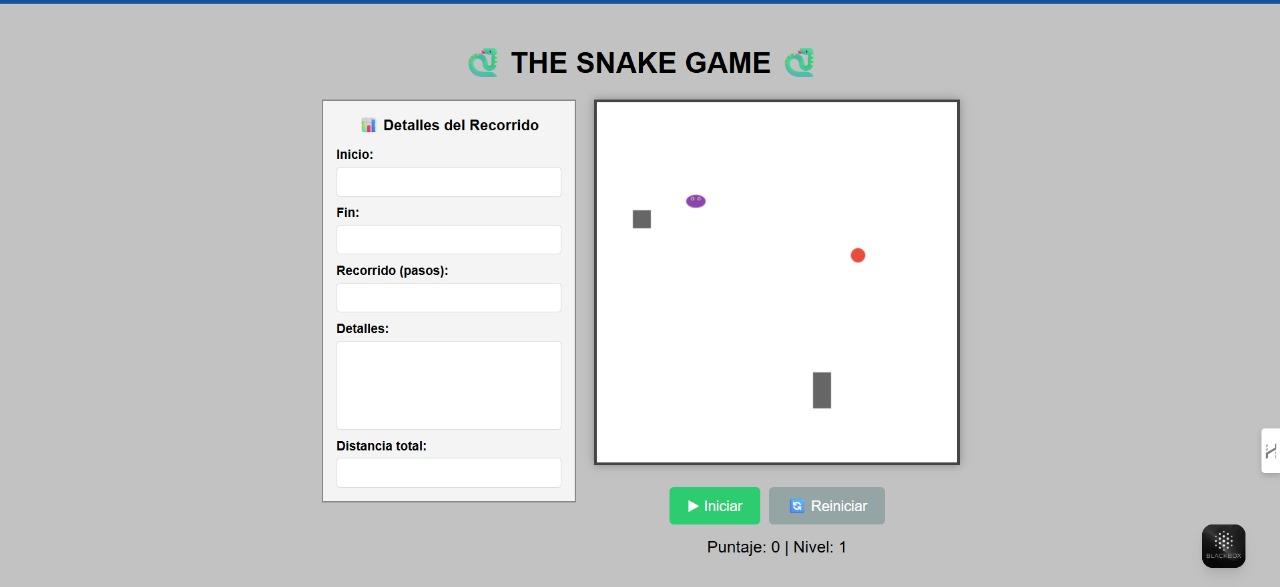
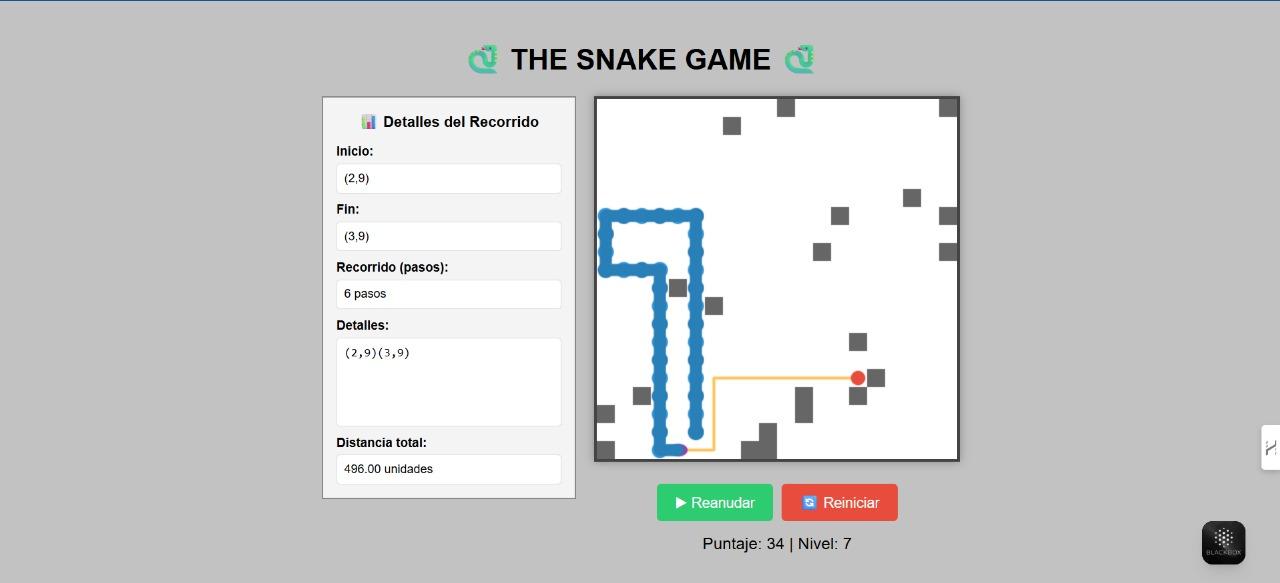


Figura. 2. Juego de la serpiente en inicialización.

Figura 3. Juego de la serpiente en ejecución, se muestra el camino más corto hacia su alimento.

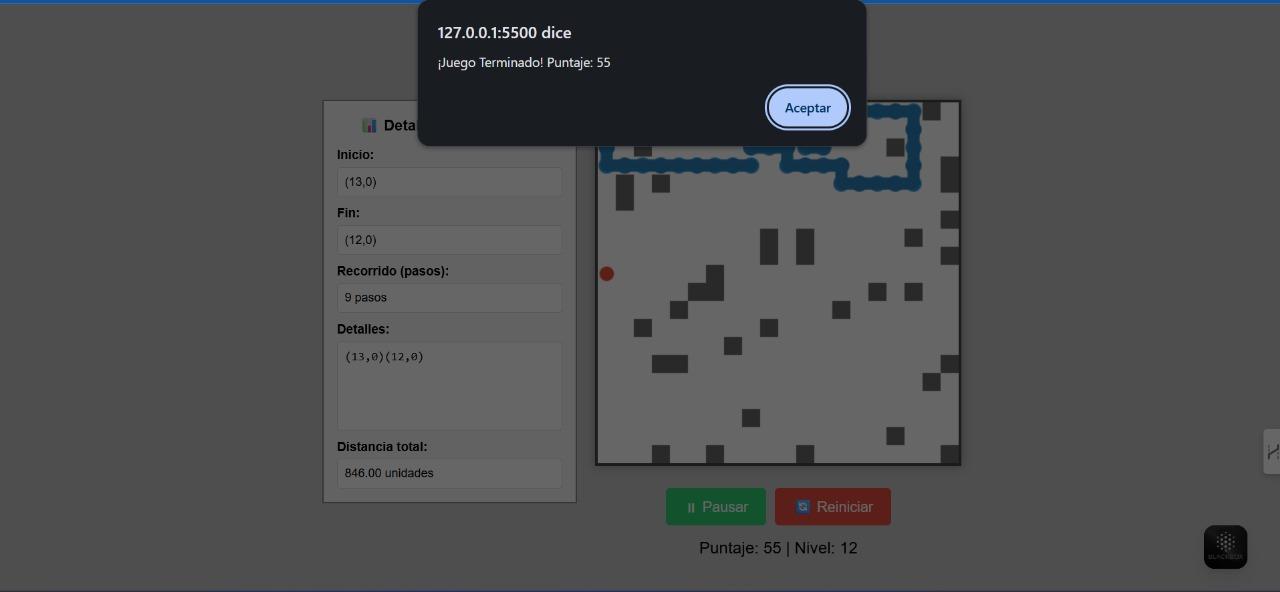
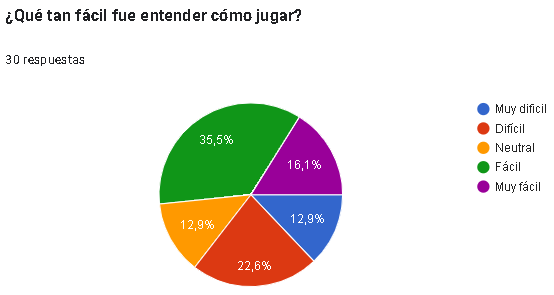
**

Figura. 4 Juego finalizado, aquí la serpiente se encontró ante un obstáculo que no logró evitar y con esa acción finaliza la ejecución del juego

Figura 5.Ilustración gráfica de la calificación de la complejidad del juego

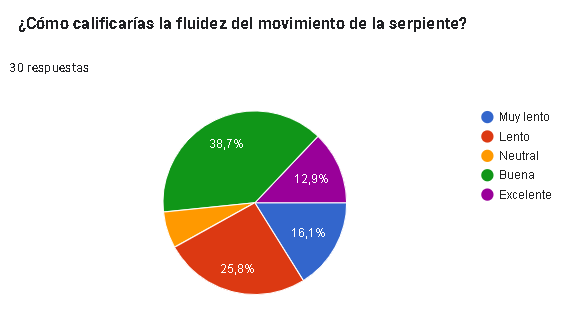


Figura 6. Ilustración gráfica de la calificación de la fluidez del juego

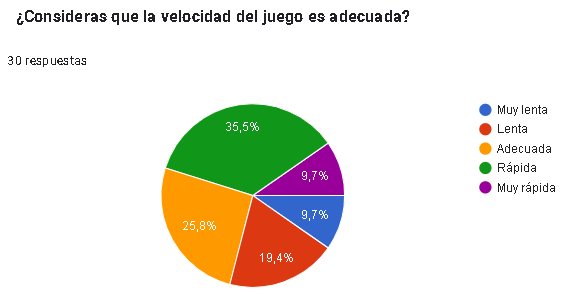


Figura 7.Ilustración gráfica de la calificación de la velocidad del juego



Figura 8. Ilustración gráfica de la calificación del diseño del juego

### 

Figura. 9. Ilustración gráfica de la calificación de colisiones o errores del juego.

### 

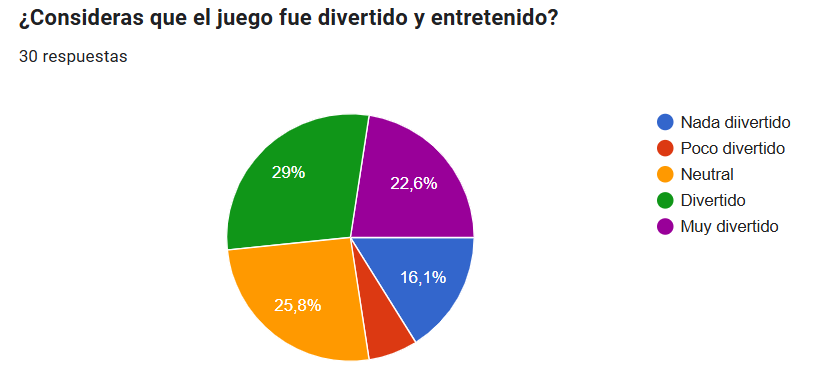


Figura 10. Ilustración gráfica de la calificación de satisfacción del juego.

### G. Comparación con variantes y justificación del enfoque

Aunque existen alternativas como A\*, Dijkstra o versiones paralelas de BFS [2], [4], [5], se optó por el BFS clásico debido a su menor complejidad computacional, su garantía de caminos mínimos en grafos no ponderados y su implementación sencilla en JavaScript. Estudios como el de Bertolotti et al. [5] muestran que BFS puede ser una estrategia robusta incluso en entornos con compilación delta, reforzando su aplicabilidad en contextos no tradicionales como los videojuegos.

Se consideró también que BFS permite visualizar de forma clara el proceso de exploración de grafos, lo que lo convierte en una herramienta didáctica ideal para estudiantes y usuarios que buscan comprender los fundamentos de la teoría de grafos aplicada [1], [6].

Para acceder al código del juego y su demostración interactiva, se puede consultar la referencia [9]

CONCLUSIONES

El presente proyecto representó una experiencia muy valiosa tanto a nivel técnico como académico. A través del desarrollo del juego Snake utilizando el algoritmo Breadth-First Search (BFS), se aplicó de manera práctica los conceptos estudiados en teoría de grafos y algoritmos de recorrido. Modelar el tablero como un grafo dirigido y optimizar la planificación de rutas en tiempo real nos permitió comprender cómo se pueden llevar a la práctica estructuras abstractas para resolver problemas concretos.

Durante el proceso, fue necesario revisar y analizar diversos artículos científicos, lo cual fortaleció nuestra capacidad para investigar. Además, el trabajo en equipo fue fundamental para superar los desafíos que se presentaron, ya que permitió dividir tareas, compartir ideas y aprender colaborativamente.

Al realizar este proyecto se desarrolló una solución funcional y educativa, sino que también reforzó habilidades esenciales en ingeniería como la lógica computacional, la programación aplicada, la documentación técnica y la colaboración efectiva.

**Referencias.**

[1] M. del M. Ruíz Alcántara, Snake Game: Historia y evolución en dispositivos móviles [Trabajo de Fin de Grado], Universitat Oberta de Catalunya, 2019.

[2] X. Chen, Z. Xu, Y. Wu, and Q. Wu, "Heuristic algorithms for reliability estimation based on breadth-first search of a grid tree," Reliability Engineering & System Safety, vol. 232, p. 109083, Feb. 2023.

[3] J. P. Vasconez, F. Basoalto et al., "Comparison of path planning methods for robot navigation in simulated agricultural environments," Procedia Computer Science, vol. 220, pp. 1–8, 2023.

[4] K. Juneja and D. Khurana, “A Multithreaded-BFS Algorithm for Optimizing the Graph Computation in Multicore Processing System,” Procedia Computer Science, vol. 233, pp. 1–8, 2024.

[5] T. Bertolotti, W. Cazzola, and L. Favalli, "Delta-based compilation paths: A BFS-driven approach," Journal of Systems and Software, vol. 198, 2024.

[6] E. Valenzuela, H. Schaa et al., "Using search algorithm statistics for assessing maze and puzzle difficulty," Entertainment Computing, vol. 53, 2025.

[7] A. Autor y B. Autor, "Improved Non-Player Character (NPC) behavior using evolutionary algorithms," Entertainment Computing, vol. 50, 2024.

[8] Knuth, D. E.. *The art of computer programming, Volume 1: Fundamental algorithms* (3rd ed.). Addison-Wesley. 2024.

[9] J. Martos, “Snake Game with BFS – GitHub Repository,” GitHub. <https://github.com/JoseLuisMartos/TheSnakeGame>

[10] Formulario Evaluación de la Experiencia de Usuario

https://forms.gle/cpLansM3UHx3c8rL9