

Proyecto: Universo de Algoritmos

Aplicación de Autómatas y Técnicas de Programación

- **Modalidad de trabajo:** equipos de 2 o 3 estudiantes.
- **Entrega final:** lunes 9 de junio de 2025 (hora pactada con el profesor).
- **Lenguaje:** elegido por los estudiantes.
- **Entregables:**
 - Repositorio en GitHub con código versionado y evidencias de participación de cada integrante.
 - Manual de usuario y manual técnico del sistema.
 - Video explicativo en inglés (mínimo 5 minutos), donde todos los integrantes participen en cámara, con tiempos equitativos.

Módulo 1: Misión Interestelar – Resolución por Backtracking Recursivo

Desarrollar un sistema con interfaz gráfica que permita resolver un reto de exploración galáctica basado en una matriz $M \times N$ (mínimo 30×30), donde el objetivo es que una nave llegue desde una celda origen hasta una celda destino siguiendo ciertas reglas y restricciones complejas:

1. La matriz representará un mapa del universo cargado desde un archivo JSON, incluyendo:
 - Tamaño de la matriz (M y N).
 - Coordenadas de la **celda origen** y **celda destino**.
 - Posiciones de **agujeros negros** (celdas completamente bloqueadas).
 - Posiciones de **estrellas gigantes** (mínimo 5), que permiten destruir 1 agujero negro adyacente (si hay más, solo destruye uno).
 - Posiciones de **agujeros de gusano**, son túneles de una sola dirección que conectan dos celdas distantes. Se consumen al usarse.
 - Celdas que requieren un nivel de **carga mínima** para acceder (la nave empieza con cierta carga inicial, que puede cambiar durante el trayecto).
 - Celdas de **recarga de energía**, que permiten recargar X veces el valor actual de energía de la nave.
 - Cada celda tiene un valor de gasto de energía para la máquina. Cuando la nave agota su energía no puede continuar por ese camino y deberá retornar para probar un nuevo camino.
 - MatrizInicial: contiene las celdas ya definidas con la carga de energía que se gasta la nave al pasar por ellas.
2. El algoritmo debe:
 - Usar Backtracking recursivo para encontrar al menos una solución válida o informar que no existe una solución.
 - Registrar y mostrar la **cadena de pasos** seguida para llegar a la meta.
 - Validar y aplicar las restricciones de movimiento.
 - Mostrar gráficamente el universo, con íconos representando los tipos de celda (ver imagen 1).
 - Permitir ver la **animación paso a paso** de la ruta encontrada desde la interfaz.

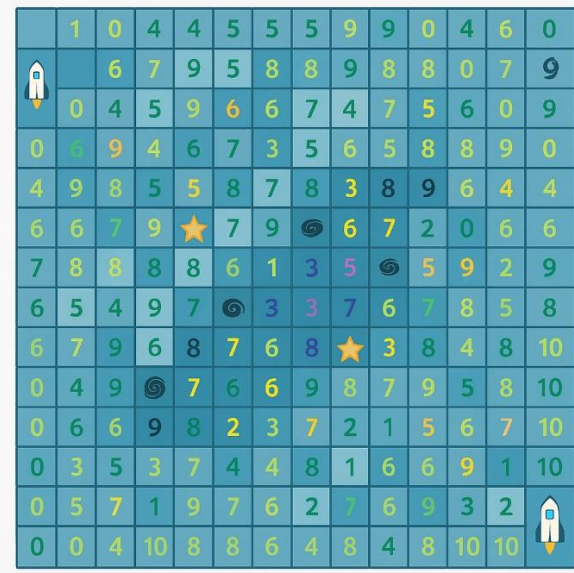


Imagen 1. Ejemplo gráfico de la matriz (solo sugerencia).

Formato de entrada JSON básico: (Adjunto al enunciado del proyecto se entrega un json de prueba que puede ser modificado según las necesidades de cada equipo).

```
{
  "matriz": { "filas": 35, "columnas": 40 }, "origen": [0, 0], "destino": [34, 39],
  "agujerosNegros": [[3,5],[10,20],[8,8]], "estrellasGigantes": [[7,7],[14,14],[20,20]],
  "portales": [{ "desde": [5,10], "hasta": [25,30] }, { "desde": [12,3], "hasta": [2,39] }],
  "agujerosGusano": [{ "entrada": [11,11], "salida": [13,13] }, { "entrada": [18,5], "salida": [21,6] }],
  "zonasRecarga": [[4,4, 3],[15,15, 2]], "celdasCargaRequerida": [{ "coordenada": [9,9], "cargaGastada": 30 }, { "coordenada": [22,22], "cargaGastada": 22 }], "cargaInicial": 200,
  "matrizInicial": [[...],[...],[...],..., [...]]
}
```

En zonas de recarga se tendrán listas de 3 posiciones, donde la primera posición es la coordenada en X, la segunda es la coordenada en Y y la tercera es la cantidad de veces por la que se multiplica la carga actual de la nave. En dichas celdas no se aplica el gasto de energía que tenga asignada la celda.

La matriz inicial contiene la matriz NxM con las celdas llenas de números entre 1 y 10, lo que representa el gasto de energía que tiene la nave al pasar por dicha celda (el valor asignado se debe disminuir a la energía de la nave).

Módulo 2 – “Sintaxis Galáctica: Un autómata para validar el lenguaje Neutrino”

Se debe diseñar e implementar un autómata finito determinista (AFD) para validar la sintaxis de un lenguaje llamado Neutrino. Este lenguaje fue creado especialmente para esta actividad, y tiene reglas más complejas que un pseudocódigo clásico.

Reglas sintácticas del lenguaje Neutrino:

- **Bloques:** deben comenzar con `iniciar` y terminar con `finalizar`.
- **Declaración de variables:**
 - `número nombre_variable;`
 - `cadena nombre_variable;`
 - `booleano nombre_variable;`
- **Asignación de valores:**
 - `nombre_variable := expresión;`
- **Operadores válidos:**
 - Aritméticos: `+`, `-`, `*`, `/`
 - Comparación: `==`, `!=`, `<`, `>`, `<=`, `>=`
 - Lógicos: `y`, `o`, `no`
- **Estructuras de control:**
 - `si condición entonces ... sino ... fin`
 - `mientras condición hacer ... fin`
 - `para nombre := valor_inicial hasta valor_final hacer ... fin`
- **Entrada/salida:**
 - `mostrar "mensaje";`
 - `leer nombre_variable;`

Reglas semánticas:

- No se pueden usar variables no declaradas.
- Cada instrucción debe finalizar con `;`.
- Las cadenas deben ir entre comillas dobles (`"`).
- El nombre de variables debe iniciar con una letra y puede incluir letras, números y guiones bajos.

Funcionalidades requeridas:

- Área de texto para cargar código desde un archivo `.txt`.
- El autómata debe:
 - Procesar línea por línea
 - Identificar errores de sintaxis
 - Mostrar número de línea, carácter del error y una descripción
- Si todo está correcto: mostrar "El código es válido".

Ejemplo de código válido:

```
iniciar
número contador;
contador := 0;
para contador := 0 hasta 10 hacer
    mostrar "Iteración";
fin
finalizar
```