



Exercici: Resolució d'un laberint amb Recurrència en Java



Objectiu

En aquesta pràctica, implementarem un programa en **Java** que resol un **laberint ASCII** mitjançant un **algorisme recursiu de IA conegut com a backtracking**.

El laberint estarà representat per una **matriu de caràcters** (`char[][]`), i el nostre objectiu serà trobar un camí des de la casella d'entrada (0,0) fins a la sortida (X).



Requisits

1. Implementar una funció **recursiva** per explorar el laberint i trobar la sortida.
2. Marcar les caselles visitades amb un caràcter ('o').
3. Fer **backtracking** si es troba un carreró sense sortida.
4. Mostrar el moviment en temps real amb una animació ASCII.



1. Representació del Laberint

El laberint és una matriu de caràcters on:

- '#' representa una paret.
- ' ' (espai) representa un camí transitable.
- 'o' representa la posició actual del nostre explorador.
- '.' representa camins visitats però descartats.



Exemple de laberint inicial:

```

  #
  # #
  #
#####
X

```



Exemple de laberint resolt:

```

o # o o o
o # o # o
o o o # o
##### o
o o o o o

```

👉 2. Implementació pas a pas

🎲 Tens mètodes predefinitos per ajudar-te. Completa les funcions que falten!

◆ 2.1 Mètodes predefinitos

```
// Mostra el laberint actual a la consola
private static void imprimirLaberint() {
    System.out.print("\033[H\033[2J"); // Esborra la pantalla per
    animació suau
    System.out.flush();
    for (char[] fila : laberint) {
        for (char c : fila) {
            System.out.print(c + " ");
        }
        System.out.println();
    }
    System.out.println();
}

// Pausa l'execució per veure la bola movent-se
private static void esperar(int milisegons) {
    try {
        Thread.sleep(milisegons);
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}
```

🎯 **Pregunta:** Per què necessitem aquestes dues funcions?

◆ 2.2 Mètode `esValid()`

🎲 Aquest mètode comprova si una casella (`x`, `y`) és transitable.

🔧 **COMPLETA EL CODI:**

```
private static boolean esValid(int x, int y) {
    return _____;
}
```

Pistes:

1. Ha de comprovar que (`x`, `y`) està dins dels **límits de la matriu**.
2. Ha de comprovar que la casella **no és una paret** (`#`).
3. Ha de comprovar que la casella **no ha estat visitada** (```) o sigui el final.

◆ 2.3 Mètode recursiu `resolLaberint()`

🎮 Aquest és el cor de l'algorisme. S'encarrega de:

- Comprovar si hem arribat a la **sortida**.
- **Marcar la casella actual** com visitada ('o').
- Intentar moure's en **les quatre direccions possibles**.
- Si cap moviment és vàlid, **es fa la crida recursiva**.

🔧 COMPLETA EL CODI:




```
public static boolean resolLaberint(int x, int y) {  
    // 1. Comprovar si la posició és vàlida  
  
    // 2. Comprovar si estem al final  
  
    // 3. Marquem la posició com a visitada  
    // Printem el laberint  
    // Petita pausa per veure el moviment  
  
    // 4. Intentar les quatre direccions possible (Dreta, Avall,  
    // Esquerra i Amunt. Per cadascun d'ells retornem true.  
  
    // 5. Si no trobem camí, tirem enrere  
    // Marquem el camí com a visitat '.'  
    // Printem el laberint  
    // Petita pausa per veure el moviment  
  
    // Si no hem retornat true fins aquí, llavors farem return false;  
}
```

🔧 3. Executant el Programa

Cridem `resolLaberint(0, 0)` des del main:

```
public static void main(String[] args) {  
    if (resolLaberint(0, 0)) {  
        System.out.println("Camí trobat!");  
    } else {  
        System.out.println("No hi ha solució");  
    }  
}
```

Extensions per a experts

-  Implementa **moviments diagonals**.
-  Fes que el laberint es generi **aleatòriament**.
-  Converteix-ho en una aplicació **gràfica** amb **Swing** o **JavaFX**.

 Ara et toca a tu!