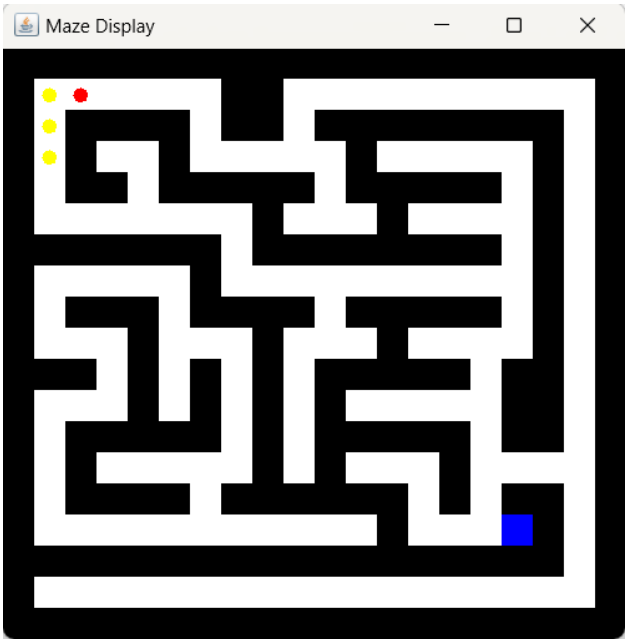
# Backtracking: MazeSolver




Der rote Punkt ist ein Spieler, der den Weg durchs Labyrinth zum Ziel (=blaues Quadrat) finden soll.

Besuchte Felder werden gelb markiert.

Zu implementieren ist die Methode `findGoal()` der Klasse `MazeSolver.java`

In der Klasse `MazeSolver` gibt es ein Attribut `maze` vom Typ `Maze`.

Im Konstruktor von MazeSolver kann man die Startwerte festlegen - und die Wartezeit zwischen den Zügen

In der Liste `currentDirections` kann man speichern, welche Züge der Player gemacht hat.

In `bestDirections` kann der (bisher) kürzeste Weg gespeichert werden;
der Weg in `bestDirections` wird Ende abgegangen. (Voraussetzung: Der Player steht am Ende wieder am Start!!)

In `bestSteps` wird die Anzahl der Schritte für den kürzesten Weg gespeichert.

Für die Klasse `Maze` gilt das Klassendiagramm unten.
Im Klassendiagramm sind nur die für die Implementierung wesentlichen Methoden aufgeführt.

| Maze |
|---|
| // Attribute spielen keine Rolle |
| + Maze(field: int[][])<br>+ isPlayerOnGoal(): boolean // prüft, ob der Spieler auf dem Ziel steht.<br>+ isPlayerOnVisited(): boolean // prüft, ob der Spieler auf einem schon // besuchten Feld steht.<br>+ isWallInDirection(direction: int): boolean // prüft, ob in der Richtung eine Wand ist.<br>+ markPlayerFieldAsVisited(mark: boolean) // markiert das Feld, auf dem der Spieler ist, als besucht // oder nimmt die Markierung weg.<br>+ move(direction: int): boolean // bewegt den Spieler in directon:<br>// 0: oben, 1: rechts; 2: unten; 3: links<br>// wenn die Bewegung auf eine Wand führen würde,<br>// dann passiert nichts und es wird false zurückgegeben.<br>// gibt zurück, ob der Spieler bewegt wurde.<br>+ moveBack(direction: int): boolean // bewegt den Spieler ZURÜCK<br>// d.h. für direction: 0 wird der Spieler nach unten bewegt.<br>// gibt zurück, ob der Spieler bewegt wurde. |