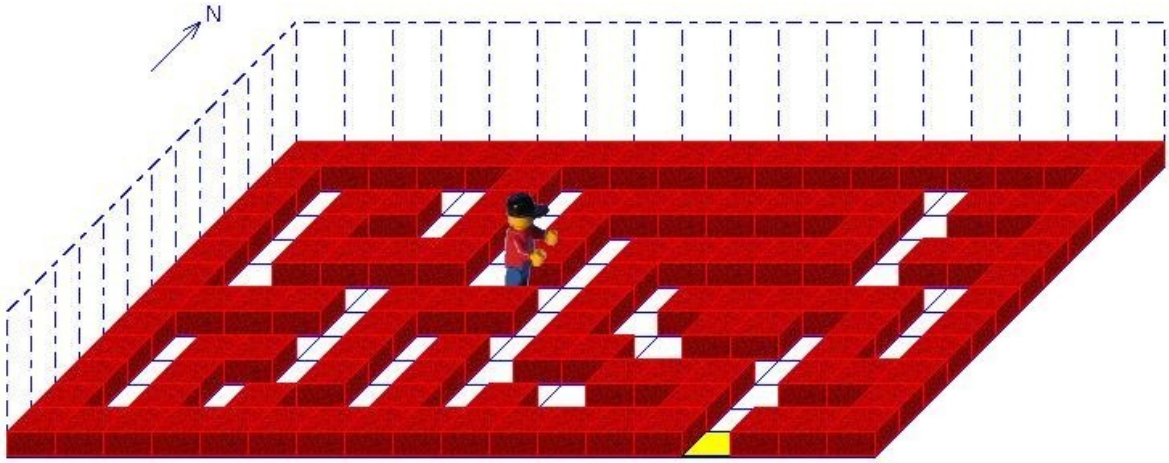
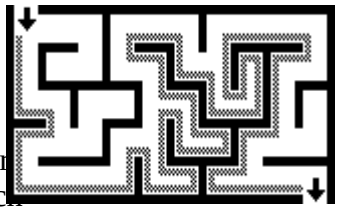
Karol im Labyrinth

Problem

Karol soll in einem solchen Labyrinth den Weg zum Ausgang finden – unabhängig von seiner Startposition und -blickrichtung und unabhängig vom genauen Verlauf der Wege. Der Ausgang ist mit einer gelben Marke gekennzeichnet. Auf dem Weg zum Ausgang soll Karol seine Schritte zählen und, wenn er das Ziel erreicht hat, ausgeben, wie weit er gelaufen ist. (Natürlich soll Karol den Wegen folgen und nicht einfach über die roten Mauern laufen.)



Lösungsansätze



Rechte-Hand-Methode: Die Rechte-Hand-Methode ist die bekannteste Regel, einen Irrgarten zu durchqueren. Man legt einfach seine rechte Hand an eine Wand des Irrgartens und hält dann beim Durchlaufen ständigen Kontakt (natürlich kann man statt der rechten auch die linke Hand verwenden). Falls alle Mauern zusammenhängen oder mit der Außenseite verbunden sind – das heißt, der Irrgarten ist „einfach zusammenhängend“ –, garantiert die Rechte-Hand-Methode, dass man entweder einen anderen Ausgang erreicht, oder wieder zum Eingang zurückkehrt.

Zufällige Wegewahl: Diese triviale Methode kann sogar von einem sehr einfachen Roboter durchgeführt werden. Sie besteht einfach darin, so lange geradeaus zu gehen, bis man eine Kreuzung erreicht. Dort entscheidet man sich zufällig für eine Richtung, in die man weitergeht.
Weil man bei dieser Methode Wege möglicherweise mehrmals beschreitet, dauert es im Allgemeinen sehr lange, bis man zum Ausgang kommt. Dennoch erreicht man diesen immer.

Quelle: de.wikipedia.org – Lösungsalgorithmen für Irrgärten, 13.10.20 (CC-by-sa-3.0, https://creativecommons.org/licenses/by-sa/3.0/deed.de)

Aufgaben

1. Überlege, wie eine Lösung der Problemstellung mithilfe der Rechte-Hand-Methode mit Javakarol umgesetzt werden kann. Entwirf ein Aktivitätsdiagramm für eine entsprechende Funktion. Dafür musst du wissen, dass `Roboter`-Objekte unter anderem über folgende Methoden verfügen: `Schritt()`, `LinksDrehen()`, `RechtsDrehen()`, `IstMarke()`, `IstZiegel()`, `IstZiegelRechts()`, `MeldungAusgeben(meldung: String)` .
2. Implementiere die Funktion und teste sie – von verschiedenen Startpositionen aus – im Labyrinth, das in der Welt-Datei KarolLabyrinth1 enthalten ist.

- Dazu musst du die Datei auf deinem Rechner speichern (Rechtsklick → Ziel speichern unter).
- In deinem Programm erzeugst du das `Welt`-Objekt dann – anders als bisher gewohnt – mit der Anweisung `Welt()`. Wenn du das Programm später startest, öffnet sich ein Datei-Auswahl-Dialog, in dem du die Welt-Datei auswählen musst.
- Die Startposition und -blickrichtung des Roboters kannst du bestimmen, wenn du bei der Erzeugung des `Roboter`-Objekts vor dem Parameter für das `Welt`-Objekt noch x-Koordinate, y-Koordinate und den ersten Buchstaben der Himmelsrichtung in einfachen Anführungszeichen angibst, zum Beispiel `Roboter(2, 2, 'N', meineWelt)`.
- Wenn du die Methode `MeldungAusgeben(meldung: String)` deines `Roboter`-Objekts benutzen willst, um die Anzahl der gelaufenen Schritte auszugeben, hast du das Problem, dass eine Zeichenkette als Meldung erwartet wird. Nehmen wir an, dass du die Anzahl der Schritte in einer Variablen mit dem Bezeichner `schritte` gespeichert hast. Dann kannst du als Zeichenkette für die Meldung schreiben: `"Anzahl Schritte: $schritte"`. Das bewirkt, dass der Variablenwert von `schritte` an der mit dem Dollarzeichen bezeichneten Stelle in die Zeichenkette eingefügt wird.

3. Entwirf ein Aktivitätsdiagramm für eine Lösung der Problemstellung durch zufällige Wegewahl. Dazu musst du wissen, dass es in Kotlin ein Objekt mit dem Bezeichner `Random` gibt, das Zufallszahlen erzeugen kann. Damit du es benutzen kannst, schreibst du in der obersten Zeile deines Programms Folgendes: `import kotlin.random.Random`. Nun kannst du weiter unten die Methode `nextInt(maximumExklusiv: Int)` des Objekts benutzen: Zum Beispiel liefert dir der Methodenaufruf `Random.nextInt(3)` zufällig eine `0`, `1` oder `2`.
4. Implementiere deine Funktion und teste sie wie in Aufgabe 2.