TECHNISCHE UNIVERSITÄT DARMSTADT

Kapitel 01c: Fallbeispiel mit FopBot

Karsten Weihe

TECHNISCHE UNIVERSITÄT DARMSTADT

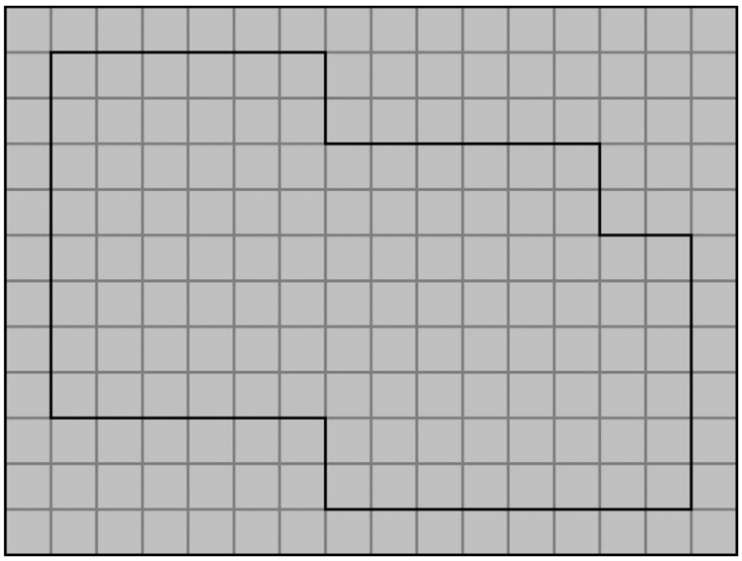
Abgegrenzten Raum mit Münzen füllen

In diesem Abschnitt sehen wir nicht viel Neues zu Java selbst, sondern ein etwas komplexeres, durchaus schon realistisches Anwendungsbeispiel für das bisher Gesehene.

Wir werden auch etwas Neues zu FopBot sehen: In der FopBot-World kann man Wände errichten. Das sind Barrikaden, die entweder über mehrere Spalten hinweg zwischen zwei Zeilen oder über mehrere Zeilen hinweg zwischen zwei Spalten stehen. Ein Roboter kann eine solche Wand nicht überwinden. Aber er kann sie erkennen, wenn er davor steht, und dadurch vermeiden.

Abgegrenzter Raum

TECHNISCHE UNIVERSITÄT DARMSTADT



In diesem Beispiel sind die Wände so platziert, dass sie so etwas wie ein geschlossenes Zimmer bilden.

Abgegrenzter Raum

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 2; i++ )
  World.placeVerticalWall ( 6, 1 + i );
for ( int i = 0; i < 6; i++ )
  World.placeVerticalWall ( 14, 1 + i );
for ( int i = 0; i < 8; i++ )
  World.placeVerticalWall ( 0, 3 + i );
for ( int i = 0; i < 2; i++ )
  World.placeVerticalWall ( 12, 7 + i );
for ( int i = 0; i < 2; i++ )
  World.placeVerticalWall ( 6, 9 + i );
```

So wie die Klasse Robot, ist auch die Klasse World im FopBot-Paket vordefiniert.

Abgegrenzter Raum

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 2; i++ )
  World.placeVerticalWall ( 6, 1 + i );
for ( int i = 0; i < 6; i++ )
  World.placeVerticalWall ( 14, 1 + i );
for ( int i = 0; i < 8; i++ )
  World.placeVerticalWall ( 0, 3 + i );
for ( int i = 0; i < 2; i++ )
  World.placeVerticalWall ( 12, 7 + i );
for ( int i = 0; i < 2; i++ )
  World.placeVerticalWall ( 6, 9 + i );
```

Bis jetzt haben wir nur Methoden gesehen, die mit einer Variable einer Klasse aufgerufen werden, zum Beispiel mit der Variable myRobot oder myRobot2 im einführenden Programmbeispiel, später hießen unsere Variablen ari und rex.

Wir nehmen hier erst einmal nur zur Kenntnis, dass es auch Methoden gibt, die statt dessen mit dem Namen der Klasse aufgerufen werden, und werden das später in Kapitel 03c unter dem Begriff *Klassenmethoden* genauer kennen lernen.

Abgegrenzter Raum

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 2; i++ )
   World.placeVerticalWall ( 6, 1 + i );
for ( int i = 0; i < 6; i++ )
   World.placeVerticalWall ( 14, 1 + i );
for ( int i = 0; i < 8; i++ )
   World.placeVerticalWall ( 0, 3 + i );
for ( int i = 0; i < 2; i++ )
   World.placeVerticalWall ( 12, 7 + i );
for ( int i = 0; i < 2; i++ )
   World.placeVerticalWall ( 6, 9 + i );
```

Der erste Parameter gibt die Spalte links von der Wand an und der zweite Parameter die blockierte Zeile. Um direkt mehrere Wände übereinander zu platzieren, erhöhen wir den zweiten Parameter mit einer for-Schleife immer um 1, bis die gewünschte Höhe erreicht ist.

Abgegrenzter Raum

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 6; i++ )
    World.placeHorizontalWall ( 1 + i, 2 );
for ( int i = 0; i < 6; i++ )
    World.placeHorizontalWall ( 1 + i, 10 );
for ( int i = 0; i < 8; i++ )
    World.placeHorizontalWall ( 7 + i, 0 );
for ( int i = 0; i < 6; i++ )
    World.placeHorizontalWall ( 7 + i, 8 );
for ( int i = 0; i < 2; i++ )
    World.placeHorizontalWall ( 13 + i, 6 );
```

Mit dieser zweiten Methode werden nun die fünf Wände in horizontaler Richtung platziert, also zwischen zwei Zeilen.

Abgegrenzter Raum

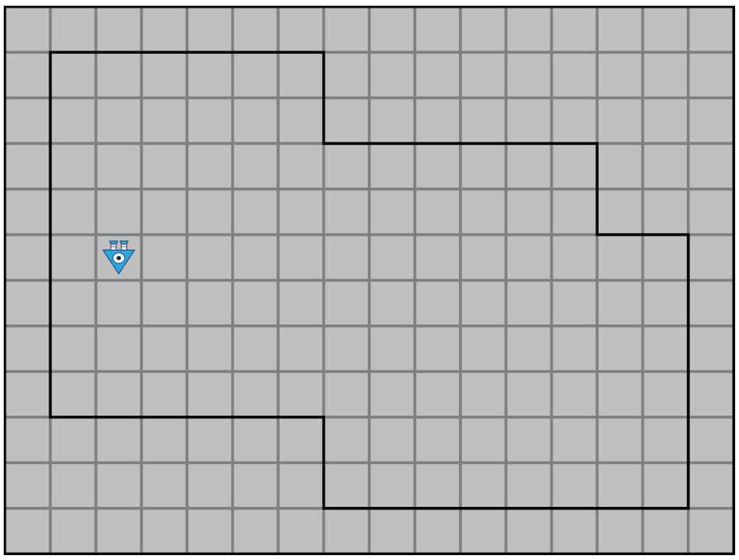
TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 6; i++ )
    World.placeHorizontalWall ( 1 + i, 2 );
for ( int i = 0; i < 6; i++ )
    World.placeHorizontalWall ( 1 + i, 10 );
for ( int i = 0; i < 8; i++ )
    World.placeHorizontalWall ( 7 + i, 0 );
for ( int i = 0; i < 6; i++ )
    World.placeHorizontalWall ( 7 + i, 8 );
for ( int i = 0; i < 2; i++ )
    World.placeHorizontalWall ( 13 + i, 6 );
```

Zu beachten ist, dass der erste Parameter hier die blockierte Spalte angibt. Der zweite Parameter ist dann die Zeile, über der die Wand verläuft, und mit einer for-Schleife platzieren wir wieder mehrere Wände nebeneinander, je nachdem wie viele Spalten insgesamt blockiert werden sollen.

Roboter irgendwo ins Zimmer

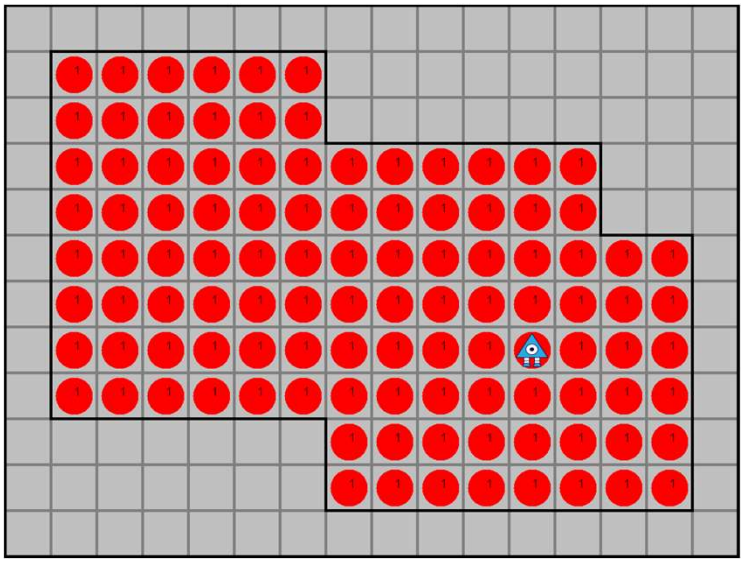
TECHNISCHE UNIVERSITÄT DARMSTADT



Der Roboter wird irgendwo in diesem Zimmer in irgendeiner Richtung platziert und weiß nicht, wie das Zimmer aussieht, also wo die einzelnen Wände stehen und wo zusammenstoßende Wände Ecken bilden. Wichtig ist nur, dass er *inner*halb und nicht *außer*halb des Zimmers platziert ist.

Aufgabe des Roboters

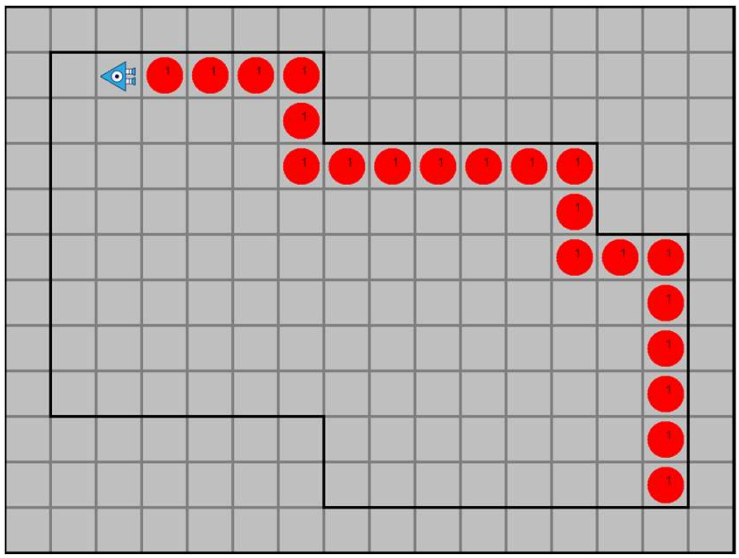
TECHNISCHE UNIVERSITÄT DARMSTADT



Die Aufgabe des Roboters ist, auf jedem Feld im Zimmer genau eine Münze abzulegen. Wo der Roboter am Ende steht und wohin er dann schaut, ist wieder egal. In stark vereinfachter Form ist das genau die Aufgabe, vor die beispielsweise Saugroboter oder Rasenmähroboter gestellt sind, nämlich eine unbekannte Fläche einmal ganz zu durchlaufen.

Vorgehensweise

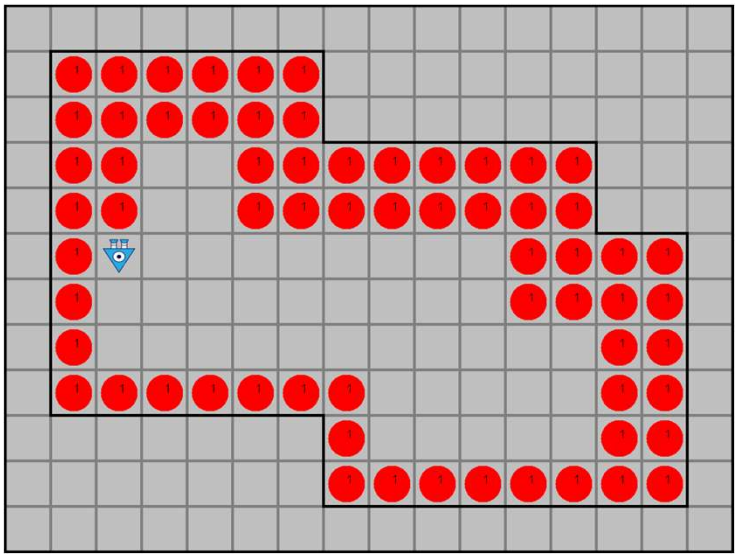
TECHNISCHE UNIVERSITÄT DARMSTADT



Die Idee ist, dass der Roboter zuerst zu einer konvexen Ecke des Zimmers läuft und dann immer an der Wand entlangläuft. Der Begriff „konvex“ bedeutet, dass die Ecke nicht in den *Raum*, sondern in das *Äußere* hineinragt, andernfalls hieße die Ecke *konkav*. Auf jedem Feld, das der Roboter durchläuft, legt er eine Münze ab. Hier sehen Sie einen Schnappschuss nach den ersten 21 Vorwärtsschritten. Startpunkt war die konvexe Ecke rechts unten.

Vorgehensweise

TECHNISCHE UNIVERSITÄT DARMSTADT



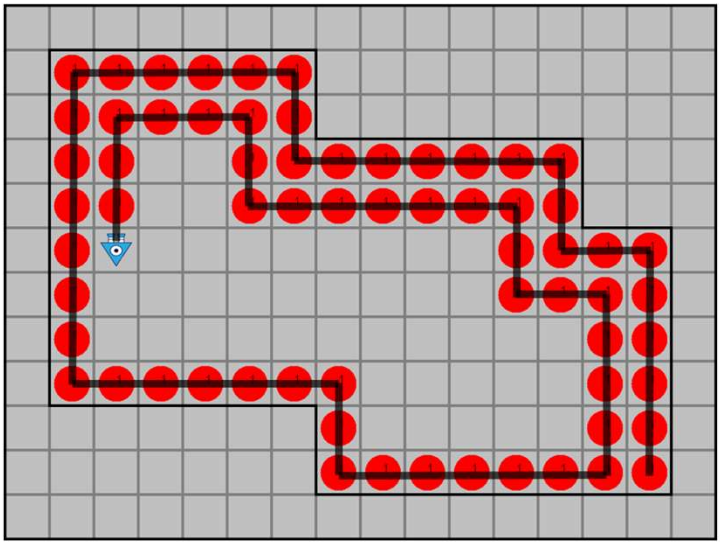
Dies ist jetzt ein zweiter Schnappschuss nach 65 Schritten.

Sobald der Roboter die ganze Wand des Zimmers entlanggelaufen ist und auf ein Feld mit Münze stößt, soll er natürlich keine weitere Münze dort ablegen, sondern sich weiter ins Innere orientieren.

Das heißt, der Roboter läuft in gewissem Sinne immer an einer Art Rand entlang, der rechts von ihm ist. Dieser Rand wird in den ersten Schritten durch Wände gebildet, in den weiteren Schritten durch Felder mit Münzen.

Vorgehensweise

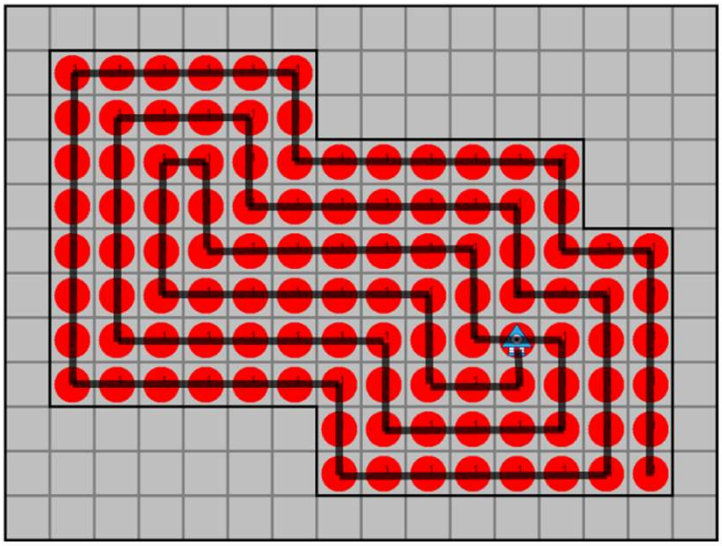
TECHNISCHE UNIVERSITÄT DARMSTADT



Das läuft darauf hinaus, dass der Roboter in einer Art Spirale immer immer weiter ins Innere des Zimmers hineinläuft.

Vorgehensweise

TECHNISCHE UNIVERSITÄT DARMSTADT

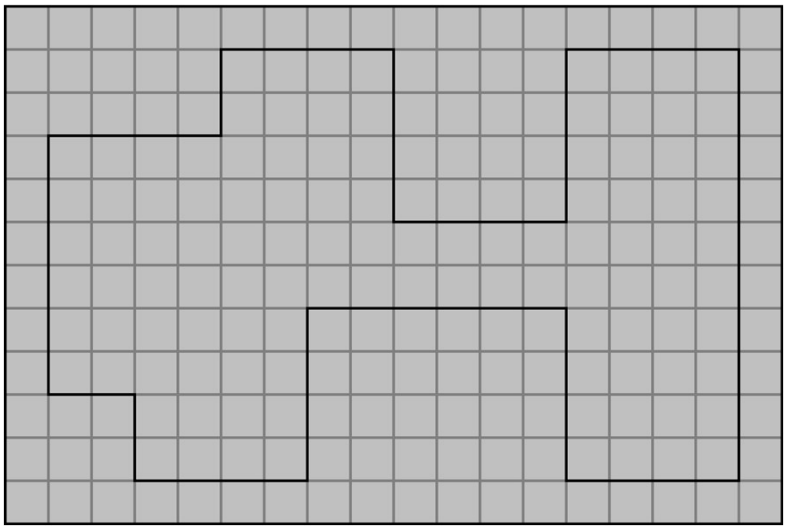


Er beendet seinen Lauf genau in dem Moment, wenn er auf kein freigebliebenes Feld mehr ziehen kann, also wenn jedes der vier benachbarten Felder jeweils durch eine Wand blockiert oder mit einer Münze besetzt ist.

Da, wo der Roboter zufällig zuletzt war, bleibt er einfach stehen. Das ist kein Problem, denn es gibt, wie gesagt, auch hier wieder keine Vorgabe, wo der Roboter am Ende stehen soll.

Nur spezielle Zimmer!

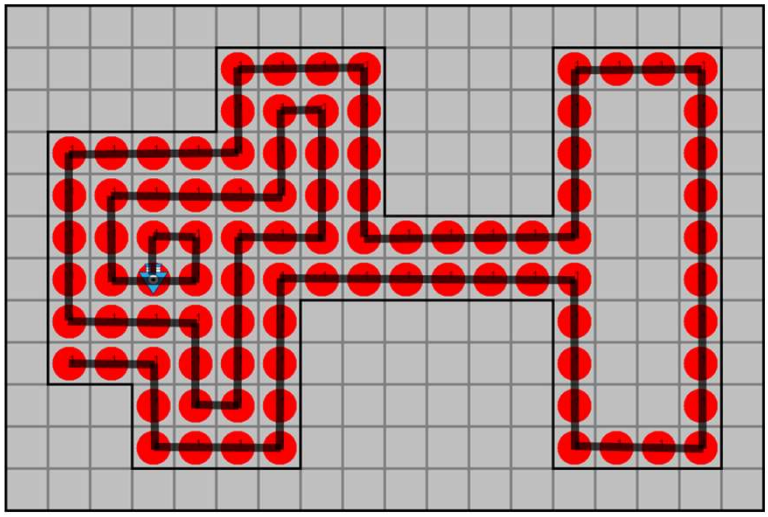
TECHNISCHE UNIVERSITÄT DARMSTADT



Der einfache Algorithmus, den Sie jetzt in Aktion gesehen haben und den wir auf den nächsten Folien implementieren werden, funktioniert allerdings nur, wenn das Zimmer gewisse Voraussetzungen erfüllt. Zunächst einmal darf es keine Engpässe wie in diesem Zimmer geben.

Nur spezielle Zimmer!

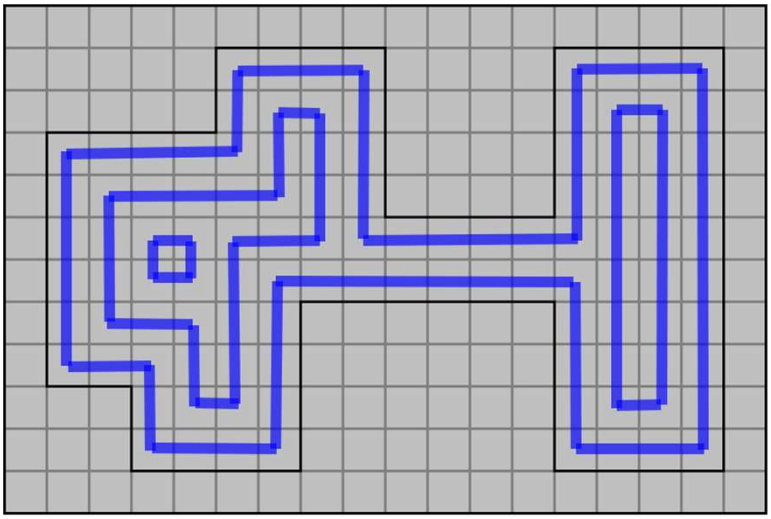
TECHNISCHE UNIVERSITÄT DARMSTADT



Sie sehen an diesem Beispiel schon das Problem: Irgendwann ist der Engpass mit Münzen blockiert, und der Roboter kommt nicht mehr auf die andere Seite hinüber, um dort noch die restlichen Felder zu füllen.

Nur spezielle Zimmer!

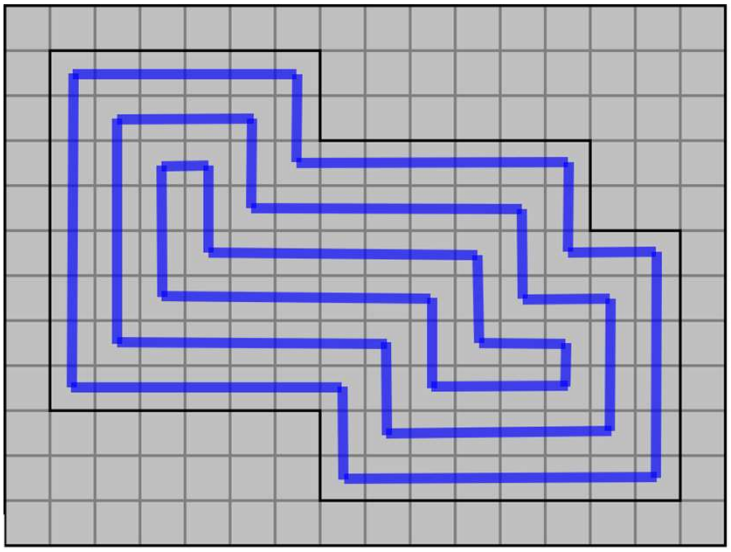
TECHNISCHE UNIVERSITÄT DARMSTADT



Begrifflich lässt sich das so fassen: Wenn wir die Felder so wie hier gezeigt nach ihrer Distanz von der Zimmerwand in Ketten zerlegen, dann ist das Problem, dass die Felder einer bestimmten Distanz nicht eine zusammenhängende Kette bilden, sondern mehrere separate. In diesem konkreten Fall zerfallen die Felder mit Distanz 2 von der Zimmerwand in zwei separate Ketten.

Nur spezielle Zimmer!

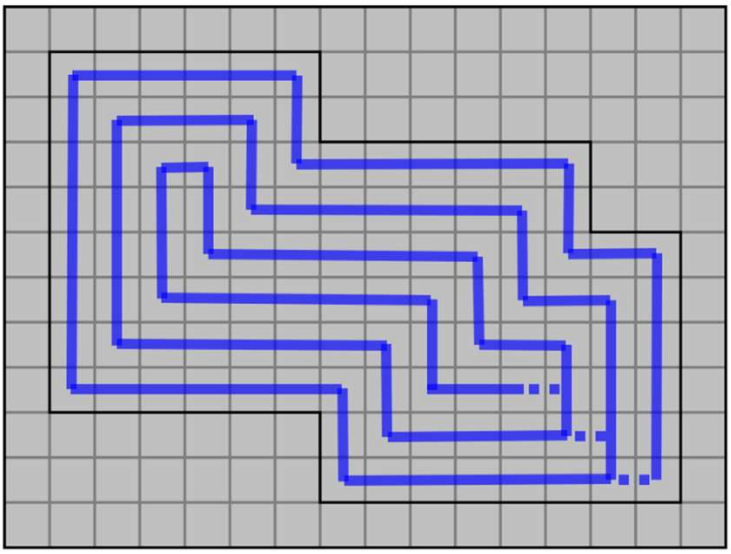
TECHNISCHE UNIVERSITÄT DARMSTADT



Die Art von Zimmern, mit denen unser Algorithmus auf jeden Fall fertig wird, sehen so aus: mehrere ineinander geschachtelte Distanzketten, so dass alle Felder derselben Distanz auf einer einzigen Kette liegen und jede Distanzkette von der jeweils nächstinneren an jeder Stelle nur einen horizontalen, vertikalen oder diagonalen Schritt entfernt ist.

Nur spezielle Zimmer!

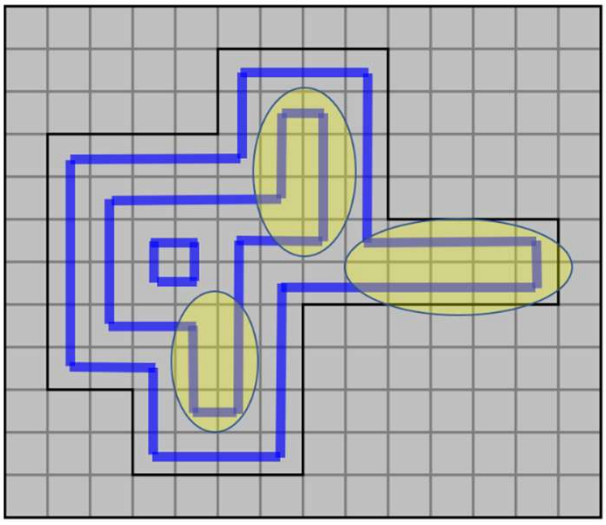
TECHNISCHE UNIVERSITÄT DARMSTADT



Denn wenn ein Zimmer diese Bedingung erfüllt, dann kann man einfach eine beliebige konvexe Ecke des Zimmers hernehmen und so wie hier gezeigt die Distanzketten zu einer Spirale umbauen, die jedes einzelne Feld im Zimmer genau einmal durchläuft.

Nur spezielle Zimmer!

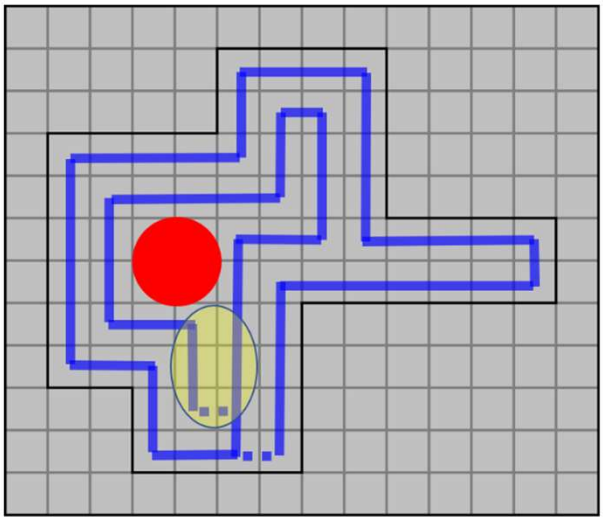
TECHNISCHE UNIVERSITÄT DARMSTADT



Hier sehen Sie ein weiteres Zimmer, das diese Bedingung *nicht* erfüllt: Der eingekreiste Abschnitt der ersten Distanzkette ganz rechts ist ein Stück von der zweite Distanzkette entfernt, und die beiden eingekreisten Abschnitte der zweiten Distanzkette sind ebenso ein Stück von der dritten Distanzkette entfernt.

Nur spezielle Zimmer!

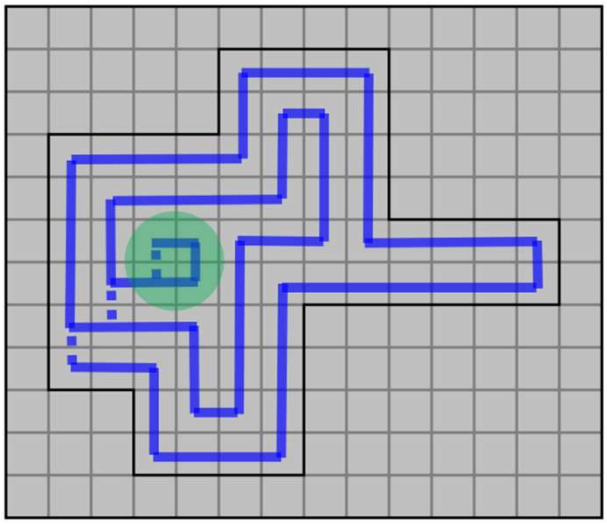
TECHNISCHE UNIVERSITÄT DARMSTADT



Startet der Durchlauf nun in einem der eingekreisten Bereiche, zum Beispiel ganz unten, dann lassen sich nur die ersten beiden Distanzketten zu einer Spirale umbauen, die dritte Distanzkette ist nicht erreichbar. Der Algorithmus, der den Roboter entlang dieser Schleife laufen lässt, wird die Felder auf der dritten Distanzkette nicht aufsuchen.

Nur spezielle Zimmer!

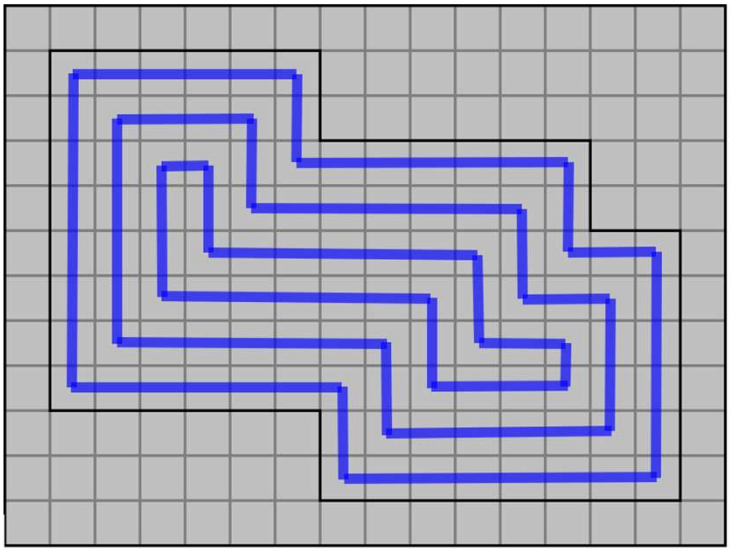
TECHNISCHE UNIVERSITÄT DARMSTADT



Würde die Ecke im selben Beispiel hingegen dort gewählt werden, wo die Bedingung lokal erfüllt ist, dann hätte der Algorithmus auch bei diesem Beispiel funktioniert.

Nur spezielle Zimmer!

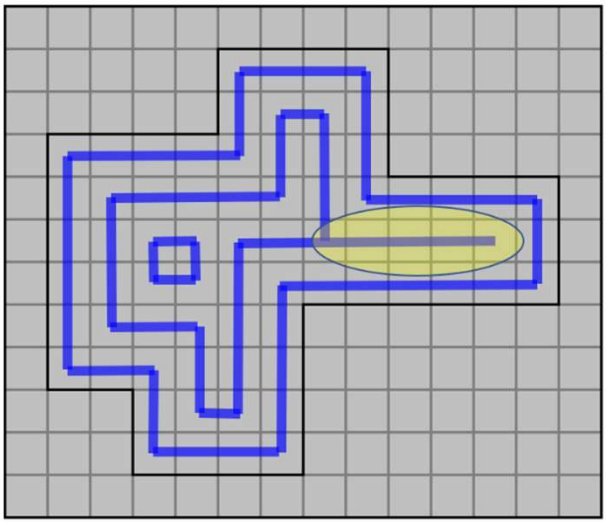
TECHNISCHE UNIVERSITÄT DARMSTADT



Da wir den Algorithmus hier simpel halten wollen, versuchen wir weder, geeignete Startecken zu finden, noch, mit Zimmern klarzukommen, in denen es keine geeigneten Startecken gibt, sondern akzeptieren, dass Zimmer, die die hier nochmals angedeutete Bedingung *nicht* erfüllen, nicht unbedingt vollständig mit Münzen gefüllt werden.

Nur spezielle Zimmer!

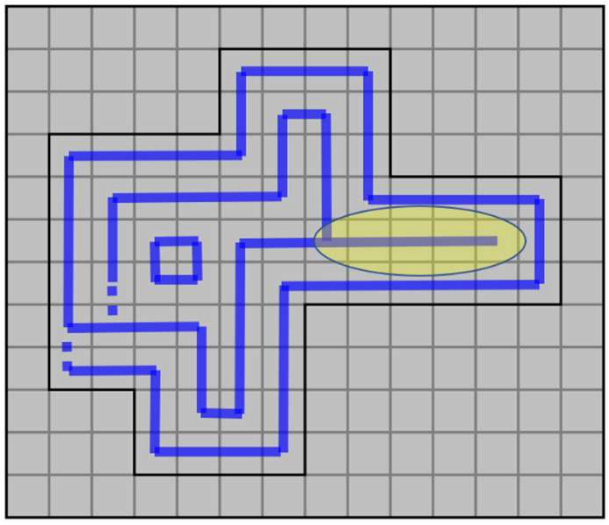
TECHNISCHE UNIVERSITÄT DARMSTADT



Noch ein Punkt ist zu beachten: Damit sich eine richtige Spirale ergibt, auf der jedes Feld genau einmal durchlaufen wird, darf keine Distanzkette quasi auf sich selbst verlaufen wie hier die zweite Distanzkette im eingekreisten Bereich.

Nur spezielle Zimmer!

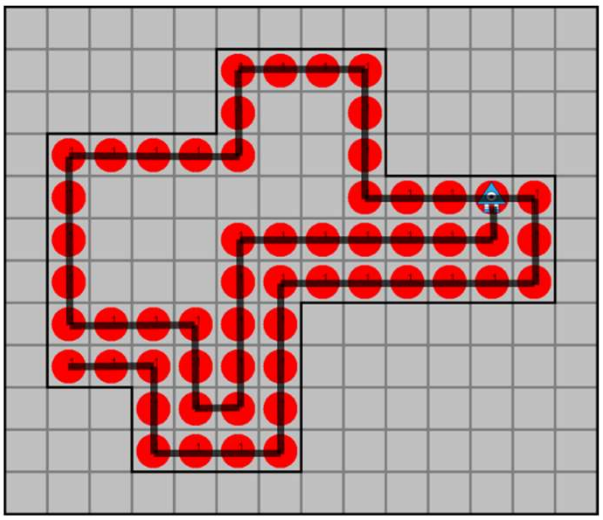
TECHNISCHE UNIVERSITÄT DARMSTADT



Denn dann funktioniert die Konstruktion der Spirale genau an dieser Stelle nicht, selbst wenn die Startecke geeignet gewählt ist.

Nur spezielle Zimmer!

TECHNISCHE UNIVERSITÄT DARMSTADT

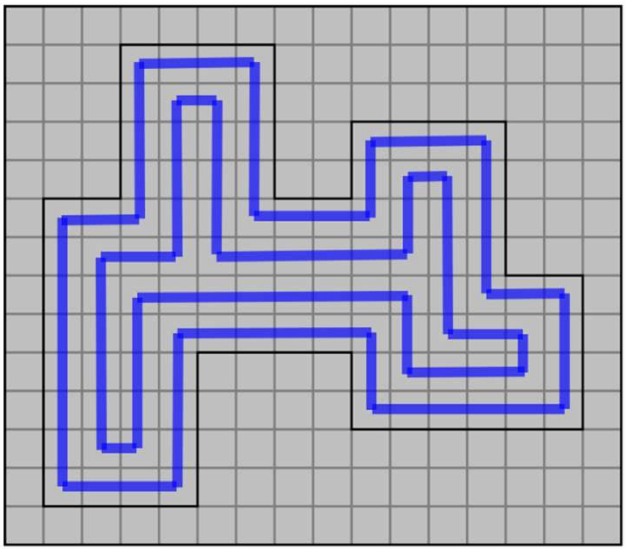


Und genau an dieser Stelle beendet dann der Roboter seinen Lauf, obwohl anderswo noch Felder unbesucht geblieben sind.

Auch solche Zimmer müssen daher von vornherein ausgeschlossen werden.

Nur spezielle Zimmer!

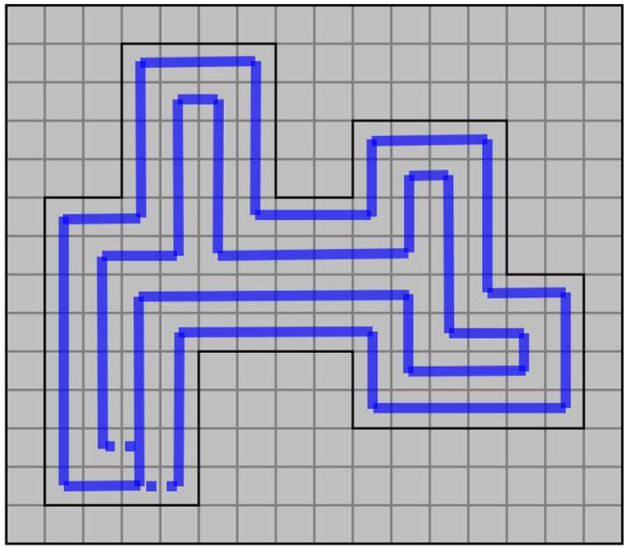
TECHNISCHE UNIVERSITÄT DARMSTADT



Bevor wir an die Implementation des Algorithmus gehen, schauen wir uns schnell noch ein weiteres, etwas komplexeres Zimmer an, das die Bedingungen ebenfalls erfüllt: Jede Distanzkette ist zusammenhängend und läuft nicht auf sich selbst zurück, und jedes Feld ist nur einen horizontalen, vertikalen oder diagonalen Schritt von der nächstinneren Distanzkette entfernt.

Nur spezielle Zimmer!

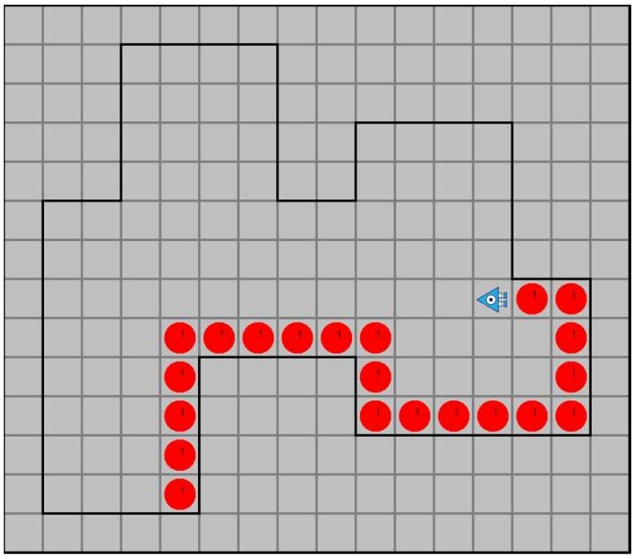
TECHNISCHE UNIVERSITÄT DARMSTADT



So dass sich auch hier von einer beliebigen konvexen Startecke aus eine Spirale konstruieren lässt, die alle Felder genau einmal durchläuft.

Nur spezielle Zimmer!

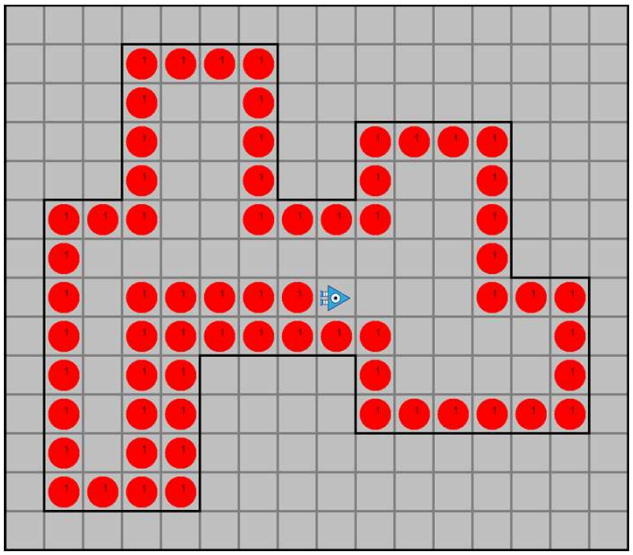
TECHNISCHE UNIVERSITÄT DARMSTADT



Schnappschuss nach 21 Schritten.

Nur spezielle Zimmer!

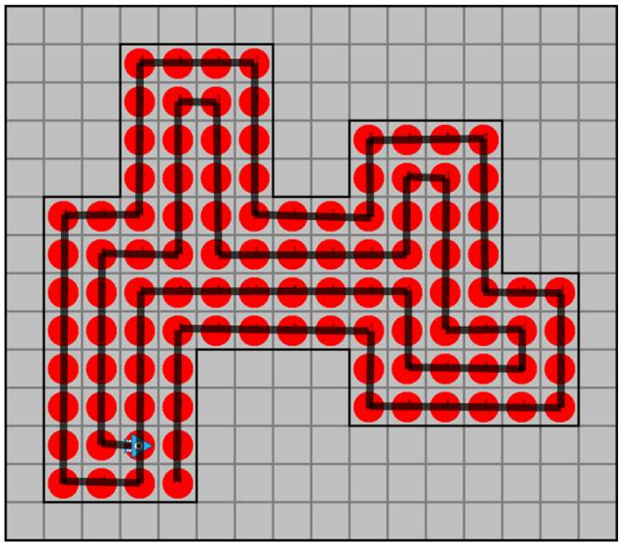
TECHNISCHE UNIVERSITÄT DARMSTADT



Schnappschuss nach 65 Schritten.

Nur spezielle Zimmer!

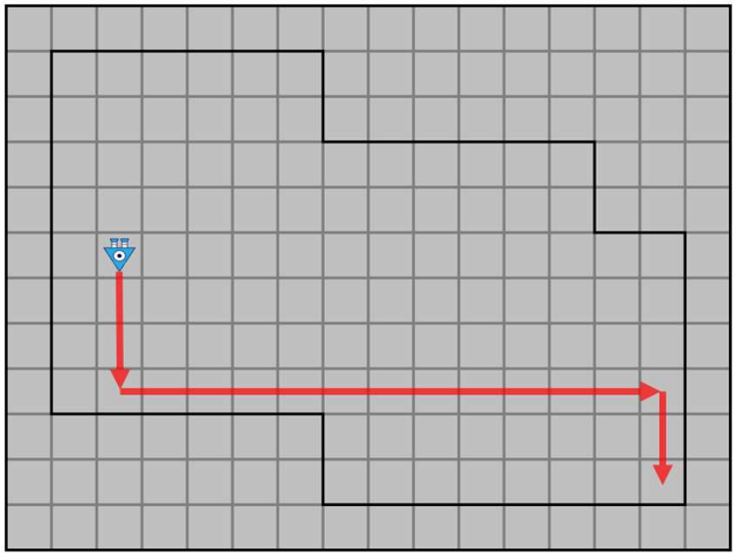
TECHNISCHE UNIVERSITÄT DARMSTADT



Und Ende des Algorithmus: Alle Felder sind genau einmal besucht worden.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT



Wir kommen jetzt zur Implementation des Algorithmus. Der erste Schritt ist, den Roboter in irgendeine konvexe Ecke zu bugsieren, egal welche. Anders gesagt, soll der Roboter zu einem Feld kommen, so dass genau neben dem Feld sowohl die Zeile als auch die Spalte jeweils in einer ihrer beiden Richtungen durch Wände blockiert sind. Die genaue Himmelsrichtung der Ecke ist egal.

Da der Roboter in diesem Beispiel zu Beginn zufällig nach unten zeigt, geht er mit der Schleife, die wir gleich implementieren werden, im Zick-Zack nach unten und nach rechts. Solange er nicht in einer konvexen Ecke gelandet ist, findet sich immer eine Möglichkeit, nach unten *oder* nach rechts weiterzugehen. Der Roboter kommt also erst dann nicht mehr weiter, wenn er dort ist, wo er hinsoll.

Würde der Roboter zu Beginn nicht nach unten, sondern beispielsweise nach links zeigen, dann wäre er im Zick-Zack nach links und unten gelaufen, analog die anderen beiden Richtungen.

Gesamtprogramm

TECHNISCHE UNIVERSITÄT DARMSTADT

!!! Zu einer konvexen Ecke !!!

!!! Roboter ausrichten !!!

!!! Spirale zum Auslegen der Münzen !!!

Wir beginnen den schrittweisen Aufbau des Programms wie üblich erst einmal mit Erinnerungsstützen. Aber wir variieren das Vorgehen leicht: Wir schreiben nicht einfach nur einmal „!!! ausfuellen !!!“, sondern machen uns schon von Anfang an etwas spezifischere Gedanken, welche Teile das Programm am Ende haben soll. Wir machen uns also im Folgenden daran, die hier stichwortartig zwischen Ausrufezeichen benannten drei Teile des Programms jeweils separat auszufüllen.

Gesamtprogramm

TECHNISCHE UNIVERSITÄT DARMSTADT

!!! Zu einer konvexen Ecke !!!

!!! Roboter ausrichten !!!

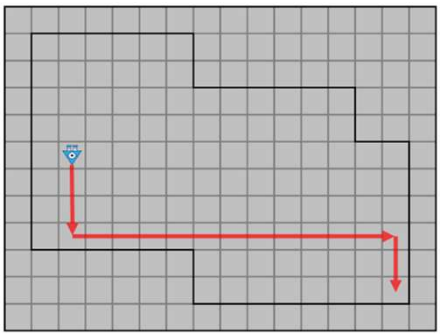
!!! Spirale zum Auslegen der Münzen !!!

Glücklicherweise ist uns dieser Punkt von Anfang an schon aufgefallen, so dass er durch diese Erinnerungsstütze von Anfang an nicht mehr vergessen werden kann: Wenn der Roboter im ersten Schritt an einer konvexen Ecke angekommen ist, schaut er ja noch nicht in die Richtung, mit der er die Spiralbewegung starten müsste, und muss daher erst noch entsprechend ausgerichtet werden.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
!!! ausfuellen !!!
while ( !!! ausfuellen !!! ) {
    if ( !!! ausfuellen !!! )
        !!! ausfuellen !!!
    else
        !!! ausfuellen !!!
}
```

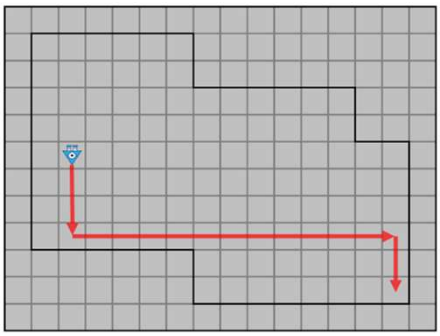


Der Code zum Navigieren zu einer Ecke besteht im Wesentlichen aus einer while-Schleife. In jedem Durchlauf durch die Schleife geht der Roboter entweder einen Schritt vorwärts oder macht eine Drehung. Das Grundgerüst können wir schon einmal hinschreiben.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
!!! ausfuellen !!!
boolean nextTurnLeft = true;
while ( !!! ausfuellen !!! ) {
    if ( !!! ausfuellen !!! )
        !!! ausfuellen !!!
    else
        !!! ausfuellen !!!
}
```



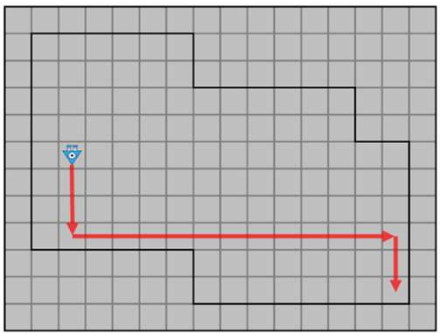
Wir wissen noch nicht so recht, wie wir das bewerkstelligen sollen, dass der Roboter immer dann, wenn er auf eine Wand läuft, abwechselnd die Richtung nach links und nach rechts ändert. Aber was wir schon vorhersehen können, ist, dass wir dafür eine boolesche Variable benötigen, in der zu jedem Zeitpunkt die Information gespeichert ist, ob die nächste Drehung nach links oder nach rechts erfolgen soll. Selbstverständlich bekommt diese Variable einen sprechenden Namen.

Letztendlich ist es egal, ob der Roboter sich als allererstes nach links oder als allererstes nach rechts drehen soll, wir entscheiden uns mit dieser Initialisierung für links.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
!!! ausfuellen !!!
boolean notFinished = true;
boolean nextTurnLeft = true;
while ( notFinished ) {
    if ( !!! ausfuellen !!! )
        !!! ausfuellen !!!
    else
        !!! ausfuellen !!!
}
```

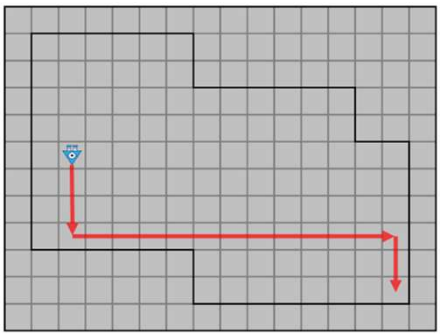


Für die Fortsetzungsbedingung wenden wir einen Trick an, der einem beim Programmieren komplexer, unübersichtlicher Schleifen häufig das Leben einfacher macht: Statt wie bisher die Fortsetzungsbedingung direkt in die runden Klammern hinter dem Schlüsselwort while hineinzuschreiben, speichern wir in einer Variable, ob die Schleife fortgesetzt werden soll oder nicht. Wenn wir dann irgendwo mitten im Schleifendurchlauf auf eine Situation stoßen, in der wir feststellen, dass wir die Schleife nach diesem Durchlauf beenden sollten, dann setzen wir dort notFinished auf false. Am Beginn des nächsten Schleifendurchlaufs wird die Fortsetzungsbedingung getestet und liefert dann false, und die Schleife ist wie gewünscht beendet.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
!!! ausfuellen !!!
boolean notFinished = true;
boolean nextTurnLeft = true;
while ( notFinished ) {
    if ( robot.isFrontClear() )
        robot.move();
    else
        !!! ausfuellen !!!
}
```



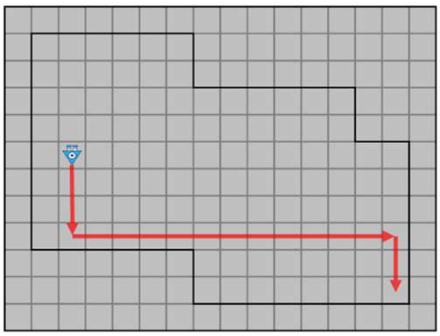
Wir schlagen in der Dokumentation zu FopBot nach, wie ein Roboter abtesten kann, ob er vor einer Wand steht. Diese Methode liefert genau dann true zurück, wenn dies nicht der Fall ist. Also wissen wir: im if-Teil der Verzweigung soll der Roboter einfach nur einen Schritt vorwärtsgehen.

Nebenbei bemerkt, haben wir uns hier jetzt den Namen für den Roboter festgelegt: einfach robot. Wir sehen bei dieser Gelegenheit noch einmal, dass Robot mit großem R und robot mit kleinem r zwei völlig unabhängige Identifier sind.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
else
    !!! ausfuellen !!!
```

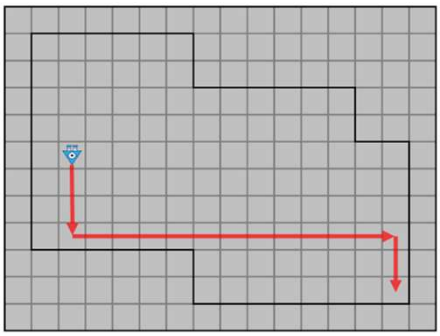


Das ist noch einmal der else-Block von der vorhergehenden Folie, also die Anweisungen, die in dem Falle ausgeführt werden, dass die if-Abfrage false liefert, also wenn der Roboter auf dem Weg zu einer Ecke vor einer Wand steht.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
else {
    !!! ausfuellen !!!
}
```

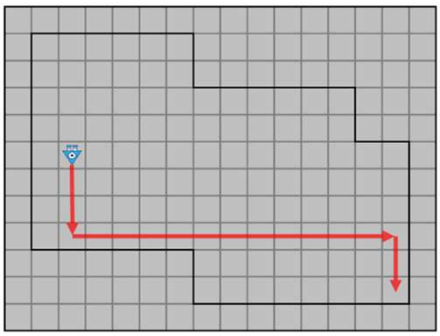


Wir schreiben gleich schon einmal geschweifte Klammern hin, damit das nicht vergessen werden kann, denn höchstwahrscheinlich haben wir hier mehr als eine Anweisung im else-Block.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
else {
    if ( nextTurnLeft )
        !!! ausfuellen !!!
    else
        !!! ausfuellen !!!
    !! ausfuellen !!
}
```

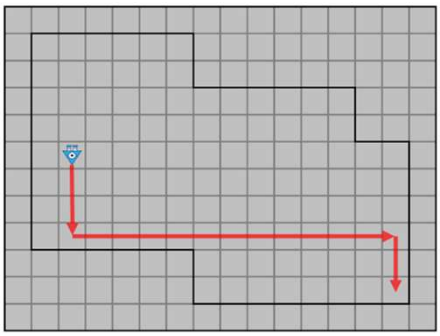


Im else-Fall muss der Roboter sich drehen, und zwar abhängig vom momentanen Wert der booleschen Variable nextTurnLeft. Ist der Wert von nextTurnLeft momentan true, dann dreht sich der Roboter nach links. Sonst dreht er sich nach rechts.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
else {
    if ( nextTurnLeft )
        robot.turnLeft();
    else
      for ( ... 3 ... )
          robot.turnLeft();
    !! ausfuellen !!
}
```

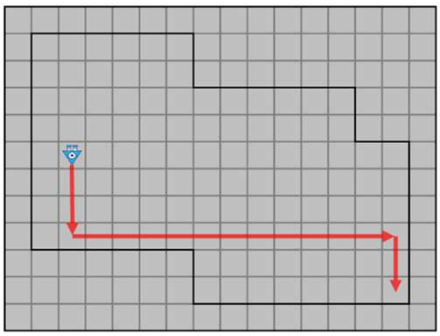


Wie das aussehen muss, wissen wir schon: Einmal nach links ist eine Linksdrehung, dreimal nach links ist eine Rechtsdrehung.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
else {
    if ( nextTurnLeft )
       robot.turnLeft();
    else
      for ( ... 3 ... )
         robot.turnLeft();
  !!! Drehrichtung aendern !!!
  !!! Abbruch der Schleife !!!
}
```

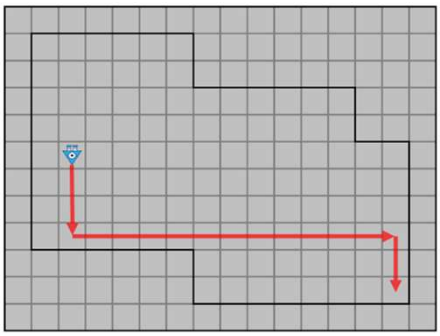


Wir müssen noch dafür sorgen, dass die Richtung der nächsten Drehung – ausgedrückt durch den Wert von nextTurnLeft – jedes Mal geändert wird, wenn der Roboter sich gedreht hat. Außerdem haben wir noch nicht den Abbruch in der konvexen Ecke implementiert.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
else {
    if ( nextTurnLeft )
       robot.turnLeft();
    else
      for ( ... 3 ... )
         robot.turnLeft();
    nextTurnLeft =
      ! nextTurnLeft;
   !! Abbruch der Schleife !!
}
```



Erinnerung aus Kapitel 01b, Abschnitt „Allgemein: Primitive Datentypen“, Logiktyp boolean: Das Ausrufezeichen dreht den Wahrheitswert des Ausdrucks, der danach kommt, um, macht also false zu true und true zu false.

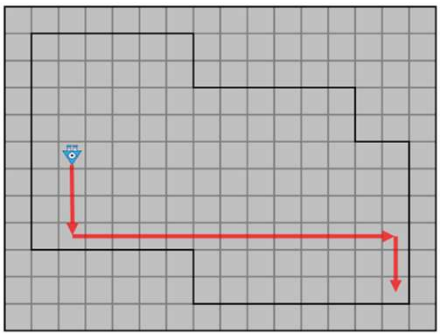
Der momentane Wert von nextTurnLeft kehrt sich also gerade um: Falls er vorher true war, wird der Variablen nextTurnLeft durch diese Zuweisung der Wert false zugewiesen; falls er vorher false war, wird ihr true zugewiesen.

Das ist auch genau richtig, denn wir wollen ja, dass sich nach jeder Drehung die Drehrichtung umkehrt.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
else {
    if ( nextTurnLeft )
       robot.turnLeft();
    else
      for ( ... 3 ... )
         robot.turnLeft();
    nextTurnLeft =
       ! nextTurnLeft;
  if ( ! robot.isFrontClear() )
     notFinished = false;
}
```



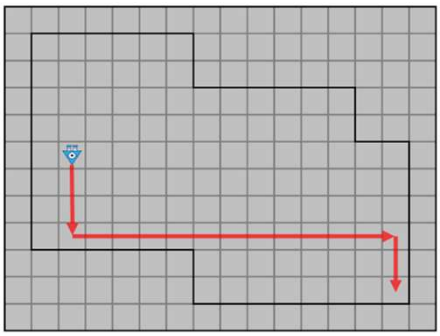
Wann ist der Roboter an einer konvexen Ecke angelangt, wie wird das erkannt? Die Erkennung ist ganz einfach: Wenn der Roboter sich dreht, weil er auf eine Wand schaut, und nach der Drehung – also an der farbig unterlegten Stelle im Programm – *wieder* auf eine Wand schaut, dann ist er in einer konvexen Ecke.

In diesem Fall setzen wir notFinished auf false. Das ist die letzte Anweisung in der Schleife, das heißt, das nächste, was danach passiert, ist das Testen der Fortsetzungsbedingung. Dieser Test liefert false. Die Schleife bricht also unmittelbar danach ab, was ja auch richtig ist, denn wir sind ja in einer konvexen Ecke angelangt.

Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

```
!!! ausfuellen !!!
boolean notFinished = true;
boolean nextTurnLeft = true;
while ( notFinished ) {
    if ( robot.isFrontClear() )
        robot.move();
    else
        ..........
}
```



Ganz vorne blieb noch eine Erinnerungsstütze stehen für den Fall, dass wir vielleicht noch etwas vergessen haben, was vor der Schleife eingefügt werden müsste. Wenn wir noch einmal die gesamte Schleife durchgehen, stellen wir fest, dass nichts mehr fehlt, so dass wir die Erinnerungsstütze löschen können.

Falls wir vergessen, noch einmal zur Erinnerungsstütze zurückzugehen und sie zu löschen, macht das nichts, denn der Compiler wird uns an selbige erinnern.

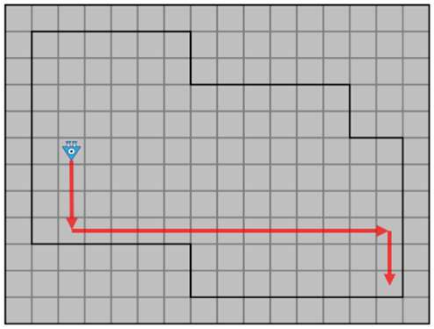
Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

***Invariante*: Der Roboter zeigt immer entweder in seine Startrichtung oder in die Richtung links davon**

***Variante*: Der Roboter geht immer in eine dieser beiden Richtungen weiter (und ändert sich in der anderen der beiden Richtungen nicht)**

***Konsequenz*: In einem geschlossenen Zimmer kommt der Roboter immer zu einer konvexen Ecke**



Wir führen jetzt zum ersten Mal die theoretischen Betrachtungen durch, mit denen man sich vergewissert, dass eine komplexe Schleife tatsächlich das tut, was sie soll. Dazu formuliert man die Invariante und die Variante der Schleife und zieht aus beidem zusammen dann die nötigen Schlussfolgerungen.

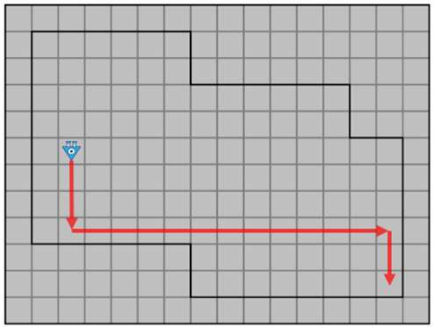
Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

***Invariante*: Der Roboter zeigt immer entweder in seine Startrichtung oder in die Richtung links davon**

***Variante*: Der Roboter geht immer in eine dieser beiden Richtungen weiter (und ändert sich in der anderen der beiden Richtungen nicht)**

***Konsequenz*: In einem geschlossenen Zimmer kommt der Roboter immer zu einer konvexen Ecke**



Die *In*variante besagt, was durch die Schleife hinweg immer gleich bleibt, eben invariant.

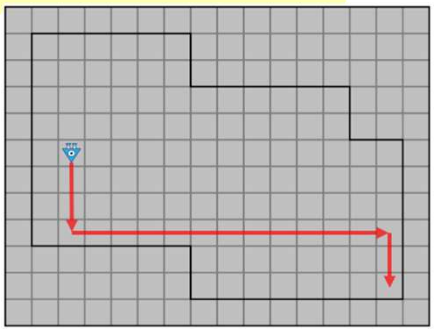
Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

***Invariante*: Der Roboter zeigt immer entweder in seine Startrichtung oder in die Richtung links davon**

***Variante*: Der Roboter geht immer in eine dieser beiden Richtungen weiter (und ändert sich in der anderen der beiden Richtungen nicht)**

***Konsequenz*: In einem geschlossenen Zimmer kommt der Roboter immer zu einer konvexen Ecke**



Die *Variante* hingegen besagt, was sich von Durchlauf zu Durchlauf ändert, also in welchem Sinne es in der Schleife vorangeht.

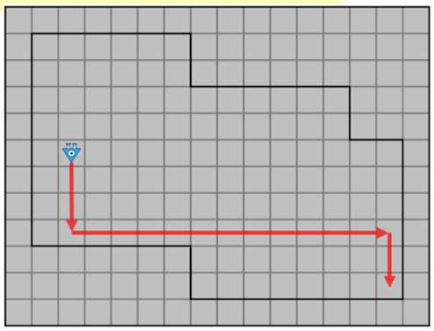
Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

***Invariante*: Der Roboter zeigt immer entweder in seine Startrichtung oder in die Richtung links davon**

***Variante*: Der Roboter geht immer in eine dieser beiden Richtungen weiter (und ändert sich in der anderen der beiden Richtungen nicht)**

***Konsequenz*: In einem geschlossenen Zimmer kommt der Roboter immer zu einer konvexen Ecke**



Dieser Punkt ist wichtig, denn aus ihm folgt, dass der Roboter in keiner der beiden Richtungen jemals zurückgeht. Wäre das nicht so, würde er also potentiell in drei oder vier Richtungen laufen, dann könnte es sein, dass der Roboter immer wieder Zykel durchläuft und nie an eine Ecke kommt.

Wir können es auch so formulieren: Die Summe aus der Anzahl Spalten plus der Anzahl Zeilen zwischen dem Roboter und der finalen Ecke nimmt in jedem Durchlauf um 1 ab und muss daher irgendwann 0 werden.

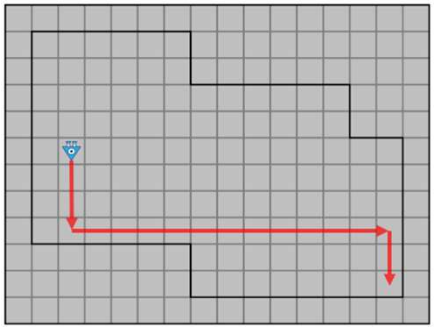
Zu einer konvexen Ecke

TECHNISCHE UNIVERSITÄT DARMSTADT

***Invariante*: Der Roboter zeigt immer entweder in seine Startrichtung oder in die Richtung links davon**

***Variante*: Der Roboter geht immer in eine dieser beiden Richtungen weiter (und ändert sich in der anderen der beiden Richtungen nicht)**

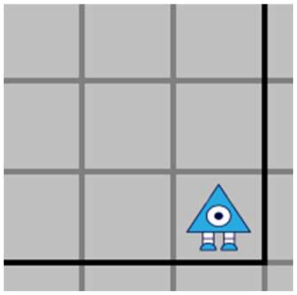
***Konsequenz*: In einem geschlossenen Zimmer kommt der Roboter immer zu einer konvexen Ecke**



Und daraus folgt dann eben, dass der Roboter nach endlich vielen Schleifendurchläufen zu einer konvexen Ecke kommen wird. Voraussetzung ist natürlich, dass der Roboter es mit einem geschlossenen Zimmer zu tun hat und nicht etwa mit einer losen Ansammlung von irgendwie platzierten Wänden, zwischen denen er hindurchschlüpfen könnte.

!!! Roboter ausrichten !!!

TECHNISCHE UNIVERSITÄT DARMSTADT



```
while ( ! robot.isFrontClear() )
    robot.turnLeft();
```

Wenn der Roboter an einer konvexen Ecke ankommt, schaut er erst einmal auf eine Wand. Um seinen Durchlauf durch das Zimmer zu beginnen, müssen wir ihn erst so drehen, dass er wie auf dem Bild eine Wand rechts neben sich und die andere Wand hinter sich hat. Das hatten wir schon von Anfang an im Auge und eine entsprechende Erinnerungsstütze in den Quelltext geschrieben, die Sie auf dieser Folie nochmals als Überschrift sehen.

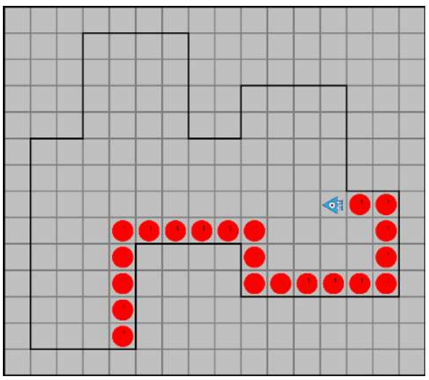
Je nachdem, auf welche der beiden Wände der Roboter schaut, müssen wir ihn also einmal oder zweimal nach links drehen. Da wir nicht wissen, ob einmal oder zweimal, lösen wir das mit einer while-Schleife, die den Roboter einmal oder zweimal dreht, eben abhängig von der Wand, auf die er schaut. Danach ist der Roboter bereit für die Schleife.

Machen wir uns noch schnell klar, dass diese Schleife zum Drehen des Robots nie endet, falls der abgegrenzte Raum aus nur einem Feld besteht. In diesem Fall verläuft die Distanzkette aber auf sich selbst, und diesen Fall hatten wir schon vorher ausgeschlossen. Daher muss dieser Fall nicht in der Methode abgefangen werden.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

```
boolean notFinished = true;
while ( notFinished ) {
   robot.putCoin();
   !!! ausfuellen !!!
}
```

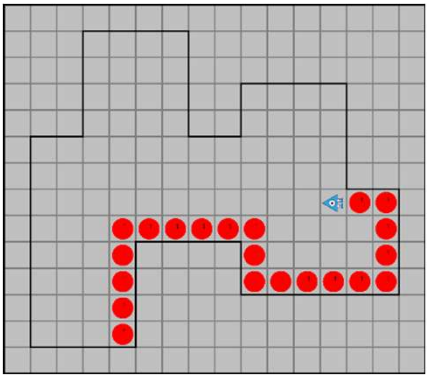


Das ist das Grundgerüst der Schleife für die Auslegung der Münzen, eine while-Schleife, die solange läuft, bis der Roboter fertig ist.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

```
boolean notFinished = true;
while ( notFinished ) {
    robot.putCoin();
    !!! ausfuellen !!!
}
```

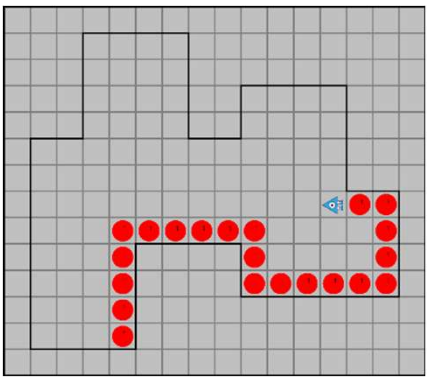


Wir hatten eben schon einmal den Trick gesehen, die Fortsetzungsbedingung in eine boolesche Variable auszulagern. Dieser Trick wird uns auch jetzt, beim Füllen des Zimmers mit Münzen, das Leben wieder leichter machen.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

```
boolean notFinished = true;
while ( notFinished ) {
   robot.putCoin();
   !!! ausfuellen !!!
}
```

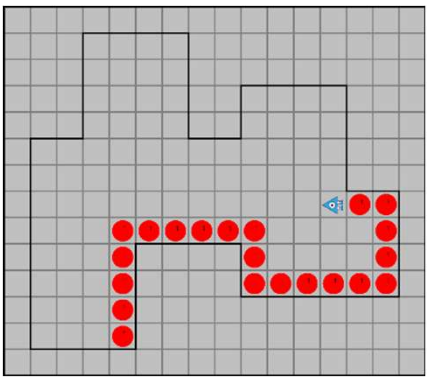


In jedem Durchlauf der while-Schleife steht der Roboter zu Beginn auf einem Feld ohne Münze, legt als erstes eine Münze ab und geht dann zum nächsten Feld auf der Spirale. Letzteres ist noch auszufüllen.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

```
boolean notFinished = true;
while ( notFinished ) {
   robot.putCoin();
   !!! ausfuellen !!!
}
```

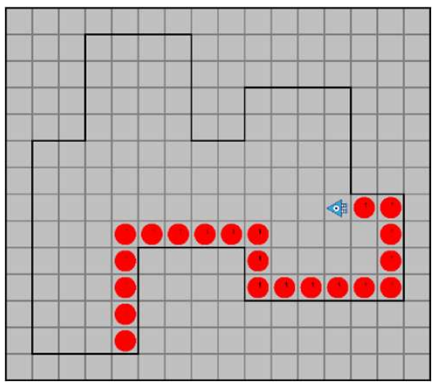


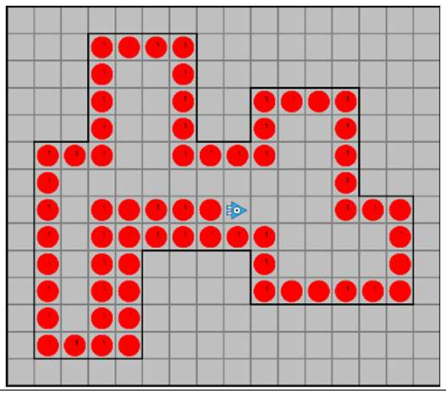
Der Schritt zum nächsten Feld auf der Spirale gestaltet sich etwas komplizierter, das passt nicht auf diese Folie. Wir schauen ihn uns auf den nächsten Folien schrittweise an. Aber zuerst vollenden wir die theoretische Betrachtung durch Formulierung von Invariante und Variante dieser Schleife.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

***Invariante*: Der Roboter hat bisher ein Anfangssegment der Spirale durchlaufen und auf jedem Feld dieses Anfangssegments eine Münze platziert (noch nicht auf seinem momentanen Feld), und er zeigt vom letzten Feld der Spirale weg**





Auch bei der Hauptschleife stellen wir dieselben theoretischen Betrachtungen an, um uns zu vergewissern, dass die Schleife korrekt ist. Diesmal gehen wir aber andersherum vor: Wir kennen die einzelnen Details der Schleife noch gar nicht, aber formulieren schon einmal Invariante, Variante und Konsequenz daraus.

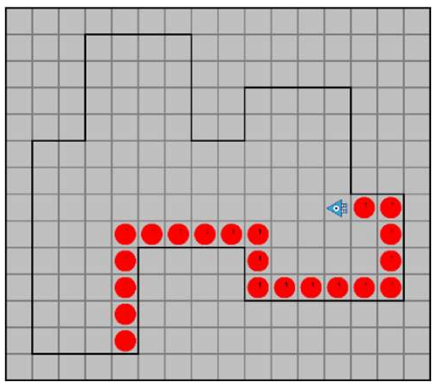
Das bedeutet, dass unser Vorgehen überlegter ist als bei der Vorschleife, in der wir den Roboter in eine konvexe Ecke manövriert hatten: erst überlegen, was die Schleife leisten soll, und daraus dann ableiten, wie sie im Details aussehen muss.

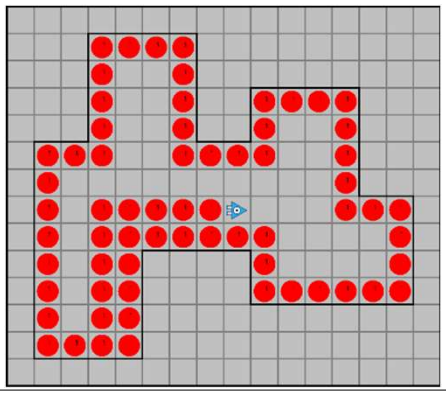
Die Invariante ist eins-zu-eins unsere Idee für das grundsätzliche Vorgehen, die wir schon herausgearbeitet hatten. Das ist durchaus typisch: Die Invariante ist in der Regel die Grundidee hinter einem als Schleife formulierten Algorithmus.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

***Invariante*: Der Roboter hat bisher ein Anfangssegment der Spirale durchlaufen und auf jedem Feld dieses Anfangssegments eine Münze platziert (noch nicht auf seinem momentanen Feld), und er zeigt vom letzten Feld der Spirale weg**





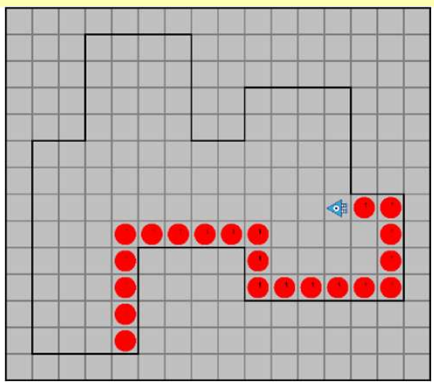
Die Invariante ist aber auch dafür da, solche unerheblich erscheinenden Punkte ein für allemal klarzustellen. Wenn wir die einzelnen Details der Hauptschleife festlegen, müssen wir entscheiden, ob wir *erst* eine Münze ablegen und *dann* vorwärtsgehen oder *erst* vorwärtsgehen und *dann* eine Münze ablegen. Beides ist gleichermaßen gut möglich, aber wir müssen uns vorab für eins von beidem entscheiden, denn wenn wir bei einzelnen Details nach der einen Version vorgehen und in anderen Details nach der anderen Version, dann wird die Schleife am Ende mit Sicherheit hochgradig fehlerhaft sein.

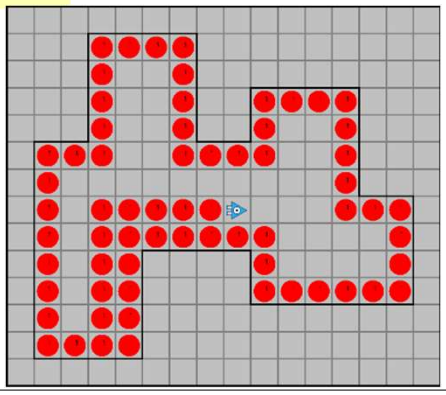
Aber auch später, wenn wir vielleicht am Algorithmus etwas ändern oder den Algorithmus erweitern wollen, ist es notwendig, die Invariante in allen Details einzuhalten. Insgesamt ist es daher sehr sinnvoll, die Invariante inklusive solcher Detailpunkte schriftlich zu fixieren – und sich beim Coden natürlich daran zu halten.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

***Invariante*: Der Roboter hat bisher ein Anfangssegment der Spirale durchlaufen und auf jedem Feld dieses Anfangssegments eine Münze platziert (noch nicht auf seinem momentanen Feld), und er zeigt vom letzten Feld der Spirale weg**



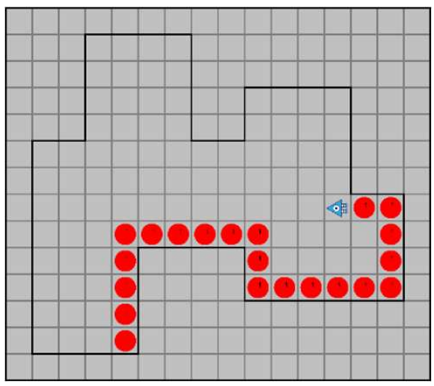


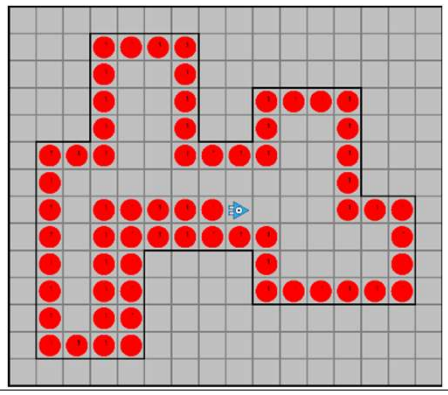
Auch solche kleinen Details sollten unbedingt schriftlich fixiert werden. Wir werden nämlich gleich sehen, dass jeder Schleifendurchlauf darauf vertraut, dass der vorhergehende Schleifendurchlauf genau dies am Ende garantiert. Wird an der Schleife später etwas geändert, dann muss darauf geachtet werden, dass dieser Punkt eingehalten wird, sonst wird die Schleife wieder mit Sicherheit fehlerhaft.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

***Variante*: Der Roboter ist ein Feld auf der Spirale weiter**





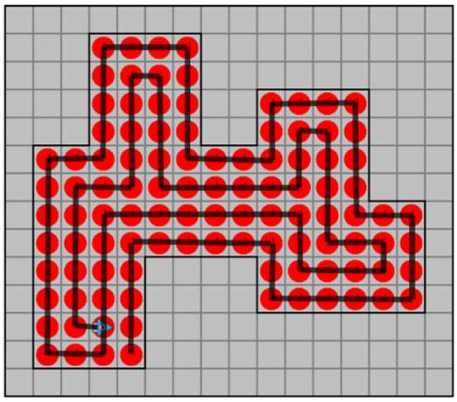
Die Variante ist in diesem Fall denkbar einfach, ist aber dennoch wichtig: Es wird ausgesagt, dass in einem Durchlauf durch die Schleife immer genau eine Münze auf einem Feld abgelegt wurde und der Roboter auf das nächste Feld auf der Spirale gegangen ist. Es gibt also keine Durchläufe, in denen der Roboter keine Münze oder zwei oder mehr Münzen abgelegt hat.

Und noch etwas kommt hinzu: Wir werden gleich sehen, dass wir die Hauptschleife nur implementieren können, wenn der Roboter auch einmal von der Spirale abweicht und andere Felder probeweise besucht. Die Variante besagt, dass der Roboter zum Ende des Schleifendurchlaufs unbedingt wieder auf der Spirale sein muss.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

***Konsequenz*: Der Roboter hat nach so vielen Durchläufen, wie die Spirale lang ist, die gesamte Spirale durchlaufen und jedes Feld im Zimmer hat genau eine Münze**

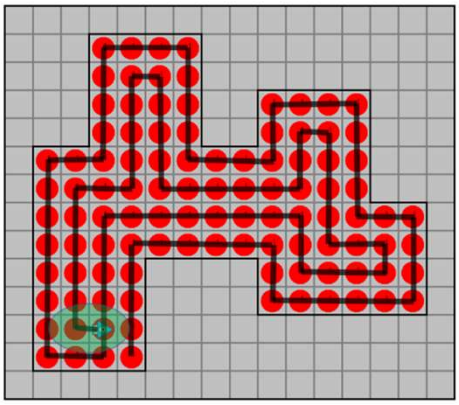


Da es nur endlich viele Felder auf der Spirale gibt, folgt aus der Variante, dass der Roboter irgendwann alle Felder durchlaufen hat. Aus der Invariante folgt, dass auf jedem dieser Felder genau eine Münze liegt.

Die Hauptschleife

TECHNISCHE UNIVERSITÄT DARMSTADT

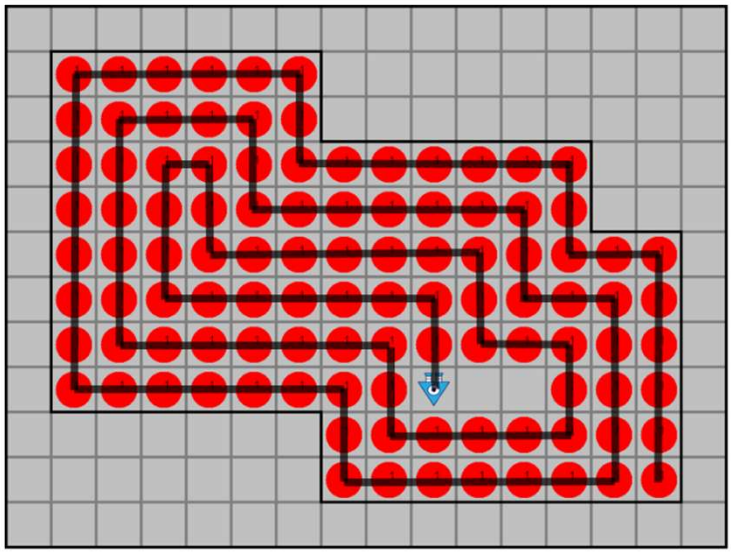
***Konsequenz*: Der Roboter hat nach so vielen Durchläufen, wie die Spirale lang ist, die gesamte Spirale durchlaufen und jedes Feld im Zimmer hat genau eine Münze**



Ein Detail ist dabei wichtig: Im allerletzten Durchlauf stellt der Roboter fest, dass er nirgendwo mehr hin kann, dass also jedes der vier benachbarten Felder entweder durch eine Wand verstellt ist oder schon eine Münze hat. In diesem letzten Durchlauf muss noch eine Münze abgelegt werden. Das wird durch die Invariante gefordert und ist auch notwendig für ein vollständig korrektes Endergebnis. Die Invariante ist also zielführend formuliert.

Die einzelnen Fälle

TECHNISCHE UNIVERSITÄT DARMSTADT

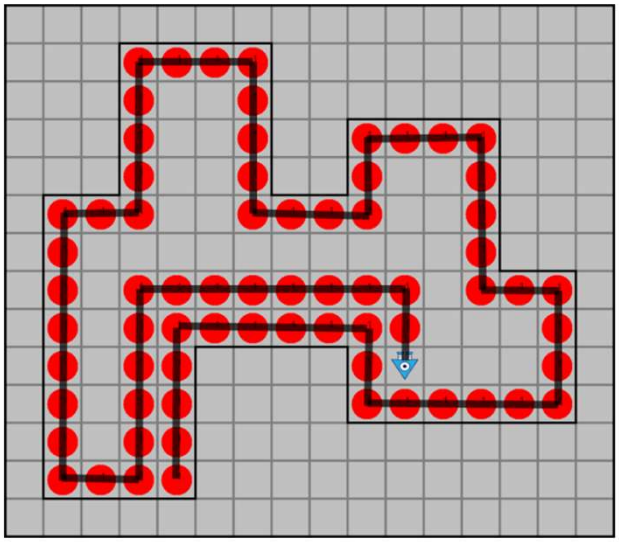


Wir haben jetzt in Invariante und Variante die wesentlichen Eigenschaften der Hauptschleife formuliert, nun implementieren wir sie.

Wir müssen verschiedene Fälle betrachten und jeweils behandeln. Es kann sein, wie in diesem Schnappschuss, dass der Roboter weder nach rechts noch geradeaus gehen soll, weil in beiden Richtungen jeweils entweder eine Wand im Weg steht oder das nächste Feld in dieser Richtung schon eine Münze hat. Dann soll der Roboter eben als nächstes nach links gehen.

Die einzelnen Fälle

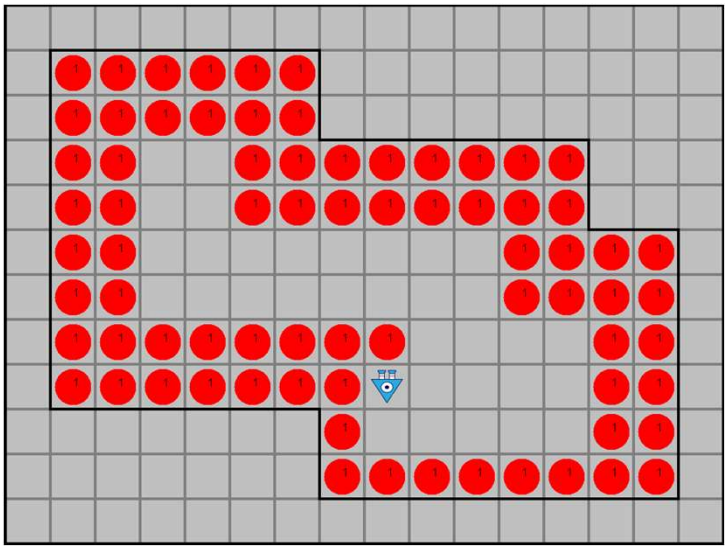
TECHNISCHE UNIVERSITÄT DARMSTADT



Dieselbe Situation im zweiten Beispiel, irgendwo mitten im Algorithmus. Auch hier geht es nur nach links weiter.

Die einzelnen Fälle

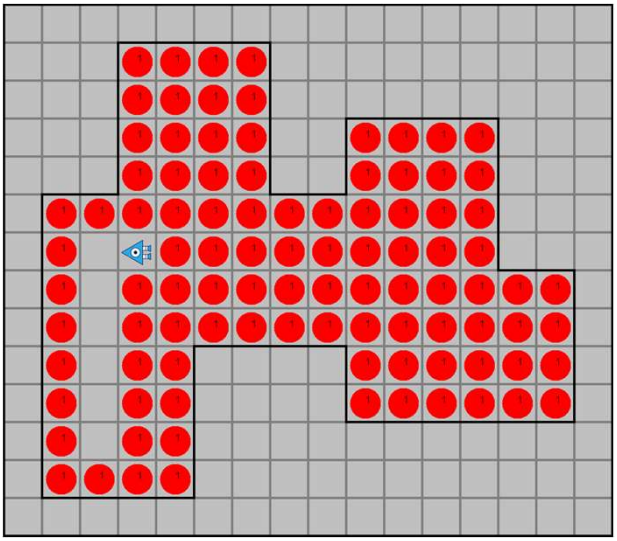
TECHNISCHE UNIVERSITÄT DARMSTADT



Oder es geht zwar rechts nicht weiter, aber in Vorwärtsrichtung ist weder eine Wand vor noch eine Münze auf dem nächsten Feld. In diesem Fall soll der Roboter ohne Drehung vorwärtsgehen. Hier wieder ein Schnappschuss vom ersten Zimmer mit diesem Fall.

Die einzelnen Fälle

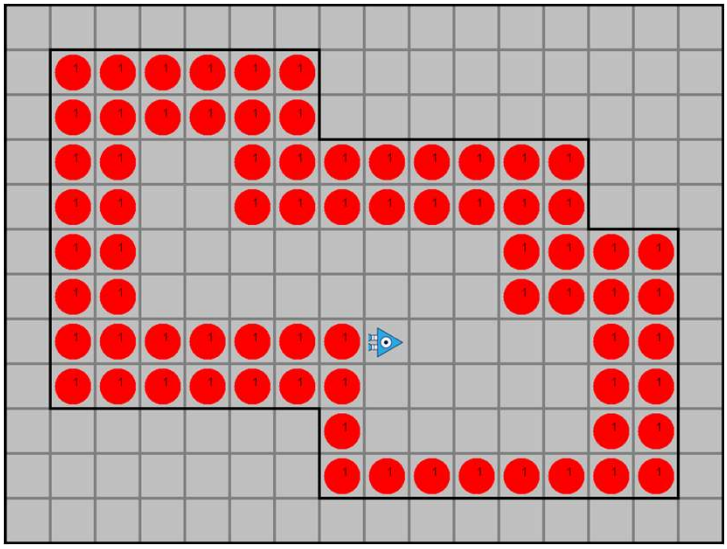
TECHNISCHE UNIVERSITÄT DARMSTADT



Und ein ebensolcher Schnappschuss vom zweiten Beispiel.

Die einzelnen Fälle

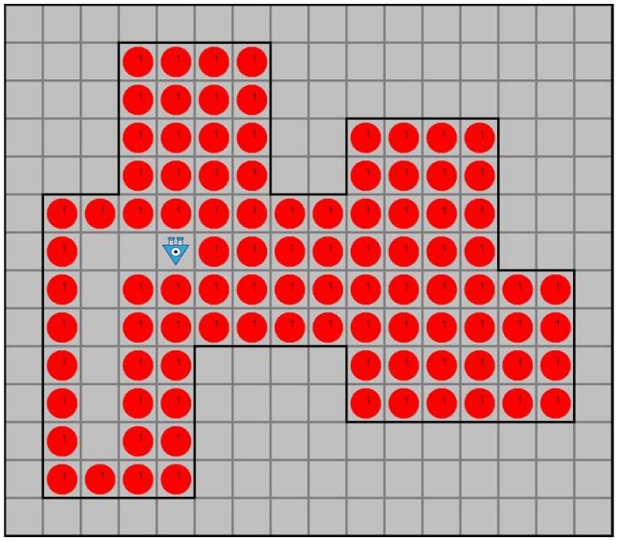
TECHNISCHE UNIVERSITÄT DARMSTADT



Und wenn es sogar rechts weitergeht, dann geht der Roboter als nächstes eben nach rechts, egal was geradeaus oder links ist. Wieder zuerst das erste Zimmer.

Die einzelnen Fälle

TECHNISCHE UNIVERSITÄT DARMSTADT

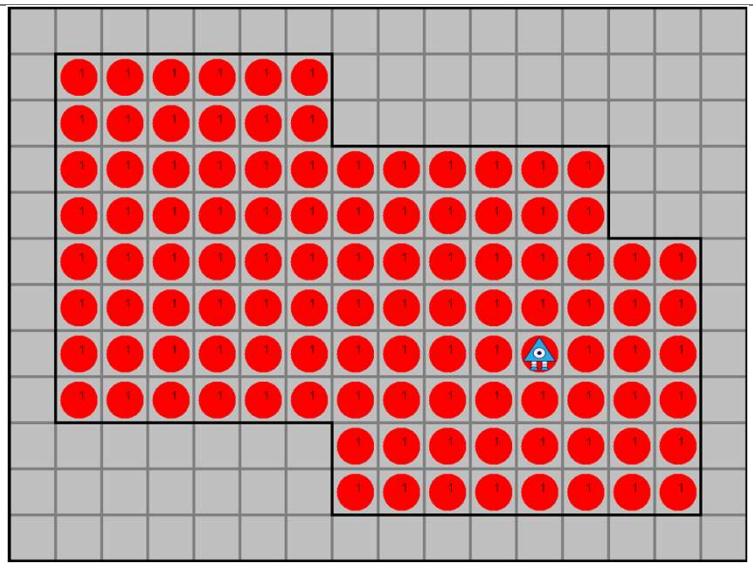


Und das zweite Zimmer.

Beachten Sie, wie wichtig es in den drei Fällen ist, dass der Roboter zu Beginn jedes Schleifendurchlaufs immer von der letzten abgelegten Münze weg zeigt. Wäre das nicht gewährleistet, dann könnten wir nicht so einfach mit den drei Fällen rechts, geradeaus und links umgehen. Gewährleistet wird dies dadurch, dass wir es explizit in die Invariante aufgenommen haben und uns bei der Implementation – und auch bei späteren Modifikationen oder Weiterentwicklungen des Codes – selbstverständlich strikt an Invariante und Variante halten.

Die einzelnen Fälle

TECHNISCHE UNIVERSITÄT DARMSTADT



Was passiert, wenn es weder nach rechts noch nach links noch geradeaus weitergeht? Dann geht es für den Roboter also in keiner Richtung mehr weiter, natürlich auch nicht zurück, denn von dort kommt der Roboter gerade her, also hat er dort im letzten Durchlauf durch die while-Schleife eine Münze abgelegt.

In diesem Fall soll die Schleife beendet sein.

Noch einmal die Schleife

TECHNISCHE UNIVERSITÄT DARMSTADT

```
boolean notFinished = true;
while ( notFinished ) {
    robot.putCoin();
    !! Roboter nach rechts !!
    !! Roboter geradeaus !!
    !! Roboter nach links !!
    notFinished = false;
}
```

Genau das ist also der Moment, in dem die boolesche Variable notFinished auf false zu setzen ist, um anzuzeigen, dass der Roboter fertig ist. Wenn wir gleich noch das vollständig implementieren, was wir hier noch mit Fülltexten bezeichnet haben, werden wir sehen, dass die Ausführung des Java-Programms zu der Zuweisung von false an notFinished nur dann kommt, wenn es in keine der drei Richtungen weitergegangen ist, und genau das ist ja auch richtig.

Diese Zuweisung ist die allerletzte Anweisung in der Schleife. Als nächstes wird die Fortsetzungsbedingung geprüft, aber jetzt liefert sie eben false, und die Schleife ist wie gewünscht beendet.

Noch einmal die Schleife

TECHNISCHE UNIVERSITÄT DARMSTADT

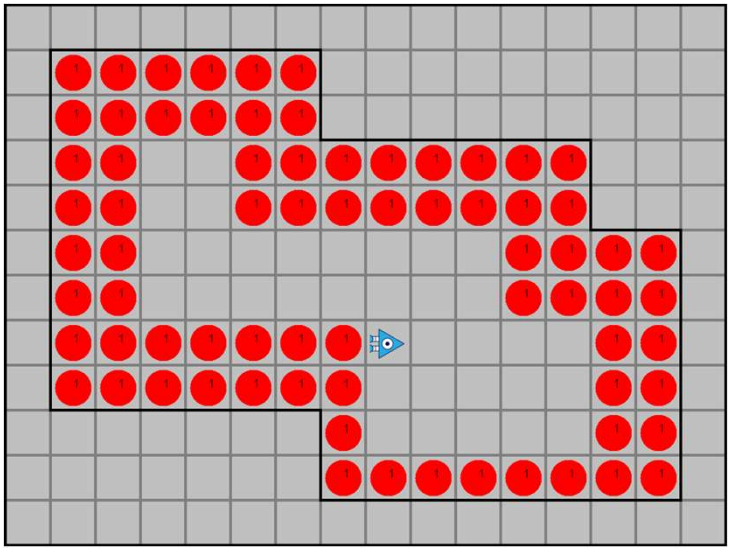
```
boolean notFinished = true;
while ( notFinished ) {
   robot.putCoin();
   !! Roboter nach rechts !!
   !! Roboter geradeaus !!
   !! Roboter nach links !!
   notFinished = false;
}
```

Es ist entscheidend wichtig, dass das Ablegen der Münze gleich am Anfang jedes Durchlaufs geschieht, denn die Invariante ist ja, dass der Roboter zwischen zwei Schleifendurchläufen auf einem Feld steht, auf dem noch keine Münze abgelegt wurde. Bevor der Roboter von diesem Feld aus weiterläuft, sollte also unbedingt eine Münze abgelegt werden.

Damit ist auch der Detailpunkt berücksichtigt, dass im allerletzten Schleifendurchlauf, in dem der Roboter feststellt, dass er nicht mehr weiterkommt, noch eine Münze abgelegt wird.

Nächster Schritt nach rechts?

TECHNISCHE UNIVERSITÄT DARMSTADT



Jetzt gehen wir daran, alle drei Fälle zu implementieren. Wir halten uns an die Reihenfolge, in der die Fälle zu berücksichtigen sind. Die erste Fallentscheidung ist dann die, ob es wie in diesem Schnappschuss rechts weitergeht, denn wenn es so ist, soll der Roboter unbedingt nach rechts gehen, egal wie die Situation links und geradeaus ist.

Nächster Schritt nach rechts?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
robot.turnLeft();
robot.turnLeft();
robot.turnLeft();
if ( robot.isFrontClear() ) {
  robot.move();
  if ( ! robot.isNextToACoin() )
    continue;
!! ausfuellen !!
```

Das ist der *gesamte* Code für den Fall, dass es nach rechts weitergeht.

Nächster Schritt nach rechts?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
robot.turnLeft();
robot.turnLeft();
robot.turnLeft();
if ( robot.isFrontClear() ) {
   robot.move();
   if ( ! robot.isNextToACoin() )
      continue;
!! ausfuellen !!
```

Damit der Roboter die Situation rechts von ihm prüfen kann, muss er sich erst einmal nach rechts wenden, was wie üblich durch drei Linksdrehungen erreicht wird.

Nächster Schritt nach rechts?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
robot.turnLeft();
robot.turnLeft();
robot.turnLeft();
if ( robot.isFrontClear() ) {
   robot.move();
   if ( ! robot.isNextToACoin() )
      continue;
!! ausfuellen !!
```

Als erstes ist die Frage zu klären, ob dort eine Wand steht oder nicht.

Nächster Schritt nach rechts?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
robot.turnLeft();
robot.turnLeft();
robot.turnLeft();
if ( robot.isFrontClear() ) {
  robot.move();
  if ( ! robot.isNextToACoin() )
     continue;
!! ausfuellen !!
```

Falls nicht, geht der Roboter einen Schritt nach rechts, um dort zu klären, ob auf dem Feld eine Münze ist.

Nächster Schritt nach rechts?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
robot.turnLeft();
robot.turnLeft();
robot.turnLeft();
if ( robot.isFrontClear() ) {
   robot.move();
   if ( ! robot.isNextToACoin() )
      continue;
!! ausfuellen !!
```

Auch die Methode isNextToACoin ist boolesch. Sie liefert genau dann true zurück, wenn auf dem Feld, auf dem der Roboter steht, mindestens eine Münze liegt. Durch das Ausrufezeichen liefert die if-Abfrage also genau dann true, wenn der Roboter auf einem Feld *ohne* Münze steht.

Nächster Schritt nach rechts?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
robot.turnLeft();
robot.turnLeft();
robot.turnLeft();
if ( robot.isFrontClear() ) {
  robot.move();
  if ( ! robot.isNextToACoin() )
    continue;
!! ausfuellen !!
```

Wenn tatsächlich keine Münze auf diesem Feld ist, dann wird diese Anweisung aufgerufen, die continue-Anweisung. Diese Anweisung haben wir schon in Kapitel 01b gesehen, Abschnitt „Weiter: Anweisungen abarbeiten“. Sie ist in Java eingebaut und beendet diesen Durchlauf der Schleife. Das heißt, es geht dann gleich mit der Auswertung der Fortsetzungsbedingung weiter, und alles, was eigentlich noch an Anweisungen in der Schleife kommt, wird übersprungen.

Das ist hier auch absolut sinnvoll, denn wenn die Ausführung des Java-Programms durch die beiden if-Abfragen hindurch bis zur continue-Anweisung kommt, ist der Roboter einen Schritt nach rechts gegangen und hat festgestellt, dass hier keine Münze ist. Also steht er genau da, wo er in diesem Durchlauf hinkommen soll, und er schaut auch noch in die richtige Richtung.

Daher ist in diesem Schleifendurchlauf in *diesem* Fall nichts mehr zu tun, und es ist absolut richtig, alles, was in diesem Schleifendurchlauf noch folgt, einfach mit continue zu überspringen.

Nächster Schritt geradeaus?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( robot.isFrontClear() ) {
  robot.move();
  if ( ! robot.isNextToACoin() )
     continue;
  robot.turnLeft();
  robot.turnLeft();
  robot.move();
  robot.turnLeft();
  robot.turnLeft();
}
robot.turnLeft();
```

Die ersten vier Zeilen auf dieser Folie sind noch einmal die letzten vier Zeilen auf der vorangegangenen Folie, die wir eben ausgiebig besprochen hatten.

Nächster Schritt geradeaus?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( robot.isFrontClear() ) {
  robot.move();
  if ( ! robot.isNextToACoin() )
     continue;
  robot.turnLeft();
  robot.turnLeft();
  robot.move();
  robot.turnLeft();
  robot.turnLeft();
}
robot.turnLeft();
```

Wenn dieses Feld nun *doch* eine Münze hat, die continue-Auweisung also *nicht* ausgeführt wird, muss der Roboter zur Ausgangskreuzung dieses Schleifendurchlaufs zurück, um von dort aus zu schauen, ob es geradeaus weitergeht. Das erledigen die drei farblich unterlegten Schritte: Wendung um 180 Grad und einen Schritt vorwärts.

Nächster Schritt geradeaus?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( robot.isFrontClear() ) {
  robot.move();
  if ( ! robot.isNextToACoin() )
     continue;
  robot.turnLeft();
  robot.turnLeft();
  robot.move();
  robot.turnLeft();
  robot.turnLeft();
}
robot.turnLeft();
```

Als nächstes ist zu prüfen, ob es in die alte Geradeausrichtung vorangeht. Dazu muss der Roboter sich nach rechts drehen, also wieder dreimal nach links.

Nächster Schritt geradeaus?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( robot.isFrontClear() ) {
   robot.move();
   if ( ! robot.isNextToACoin() )
      continue;
   robot.turnLeft();
   robot.turnLeft();
   robot.move();
   robot.turnLeft();
   robot.turnLeft();
}
robot.turnLeft();
```

Wir müssen aber auch an den Fall denken, dass der Roboter gar nicht von der Ausgangskreuzung dieses Durchlaufs nach rechts gegangen ist, weil rechts halt eine Wand steht. In diesem Fall braucht er eine einzige Linksdrehung, um wieder in die alte Vorwärtsrichtung zu schauen.

Nächster Schritt geradeaus?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( robot.isFrontClear() ) {
   robot.move();
   if ( ! robot.isNextToACoin() )
      continue;
!! ausfuellen !!
```

Und jetzt macht der Roboter genau dasselbe noch einmal wie eben. Der einzige Unterschied ist, dass der Roboter jetzt nicht mehr nach rechts schaut, von der alten Vorwärtsrichtung aus gesehen, sondern nach vorne. Im Code gibt es keinen Unterschied, die Richtung spielt keine Rolle.

Nächster Schritt geradeaus?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( robot.isFrontClear() ) {
   robot.move();
   if ( ! robot.isNextToACoin() )
      continue;
!! ausfuellen !!
```

Und auch hier wieder genau dieselbe Logik: Wenn der Roboter nicht auf eine Wand schaut und das nächste Feld in Vorwärtsrichtung noch keine Münze hat, dann bleibt der Roboter einfach auf diesem nächsten Feld mit seiner momentanen Richtung stehen, und dieser Durchlauf durch die while-Schleife kann wieder vorzeitig mit continue beendet werden. Der Roboter zeigt vom letzten Feld weg, die Invariante ist also erfüllt.

Nächster Schritt geradeaus?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( robot.isFrontClear() ) {
  robot.move();
  if ( ! robot.isNextToACoin() )
     continue;
  robot.turnLeft();
  robot.turnLeft();
  robot.move();
  robot.turnLeft();
  robot.turnLeft();
}
robot.turnLeft();
```

Und auch das, was dann weiter passiert, falls das continue *nicht* ausgeführt wird, ist wieder exakt derselbe Code: Falls der Roboter einen Schritt von der Ausgangskreuzung dieses Durchlaufs weggegangen ist, muss er wieder zu dieser zurück und sich dann nach rechts, also dreimal nach links wenden. Andernfalls war der Roboter auf der Ausgangskreuzung stehengeblieben und muss sich nur einmal nach links drehen, sonst nichts.

Nächster Schritt nach links?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( robot.isFrontClear() ) {
   robot.move();
   if ( ! robot.isNextToACoin() )
      continue;
notFinished = false;
```

Und nun im Prinzip noch einmal dasselbe Vorgehen, um auch die dritte Richtung zu prüfen.

Nächster Schritt nach links?

TECHNISCHE UNIVERSITÄT DARMSTADT

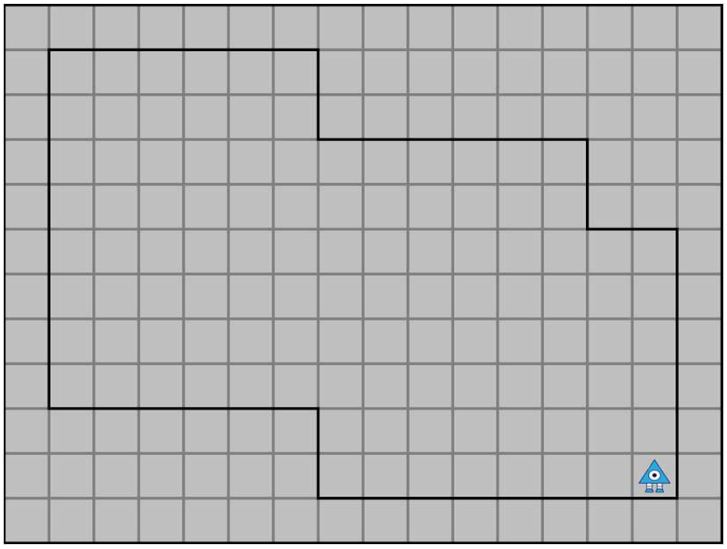
```
if ( robot.isFrontClear() ) {
   robot.move();
   if ( ! robot.isNextToACoin() )
      continue;
notFinished = false;
```

Allerdings gibt es jetzt einen Unterschied in dem Fall, dass es auch in dieser Richtung nicht mehr weitergeht. Dann sind also drei Richtungen erfolglos geprüft worden. Die vierte Richtung, aus der der Roboter im letzten Schleifendurchlauf gekommen war, braucht nicht geprüft zu werden, denn dort hatte der Roboter ja im letzten Schleifendurchlauf eine Münze abgelegt.

Es ist also genau der Fall eingetreten, in dem der Algorithmus beendet werden soll. Wie schon gesagt, ist dies die allerletzte Anweisung in der Schleife, danach wird sofort als nächstes wieder die Fortsetzungsbedingung geprüft. Diese ergibt jetzt false, und somit ist die Schleife wie gewünscht zu Ende.

Nächster Schritt nach links?

TECHNISCHE UNIVERSITÄT DARMSTADT



Wir müssen immer Ausschau nach Sonderfällen halten. In diesem Fall ist der allererste Schleifendurchlauf ein Sonderfall. Im ersten Schleifendurchlauf ist der Roboter nämlich nicht von einem Feld mit Münze gekommen. Das heißt, unser Argument, dass wir nicht in die vierte Richtung zurückschauen müssen, weil der Roboter da eh im letzten Schleifendurchlauf eine Münze abgelegt hat, stimmt beim allerersten Schleifendurchlauf nicht.

Aber beim ersten Schleifendurchlauf steht der Roboter in einer Ecke und damit im wahrsten Sinne des Wortes mit dem Rücken zur Wand. Also brauchen wir uns auch in diesem Fall nicht um die Rückrichtung zu kümmern – noch einmal Glück gehabt: zwar eine kleine Lücke in der Argumentation, warum der Algorithmus korrekt ist, aber keine Lücke im Algorithmus selbst.

Solche kleinen Lücken können bei Programmen, die Arbeiten mit hoher Verantwortung verrichten, katastrophale Konsequenzen haben, wenn sie nicht rechtzeitig entdeckt und behoben werden.