

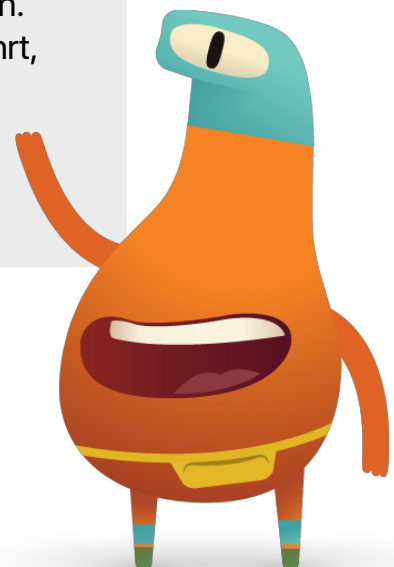
Jeder kann programmieren – Handbuch „Rätsellösungen“



Dieses Handbuch enthält Beispiele für Lösungen von Rätseln in „Programmieren lernen 1“ und „Programmieren lernen 2“. Sie werden feststellen, dass alle Rätsel mindestens ein Lösungsbeispiel enthalten und manche Rätsel eine Alternative bieten. Die alternativen Beispiele sollen verdeutlichen, dass Schüler bei der Lösung eines Problems mehrere Ansätze verfolgen können. Es ist wichtig zu beachten, dass diese Beispiele nicht die einzig möglichen Lösungen darstellen.

Einige Lösungspaare scheinen bei einem Rätsel identisch zu funktionieren. Aber sehen Sie genau hin, und Sie werden erkennen, dass es kleine Unterschiede in der Art und Weise gibt, wie sie geschrieben sind. Bei anderen Lösungen sehen Sie eine offensichtliche Änderung in der Art und Weise, wie das Rätsel gelöst wird. Der Vergleich von Lösungen – wie sie geschrieben werden und wie sie ablaufen – bietet eine ausgezeichnete Möglichkeit, Programmierkenntnisse zu verbessern.

Um eine Lösung zu testen, kopieren Sie sie einfach und fügen Sie sie in den Programmierbereich innerhalb eines Rätsels ein. Die folgenden Lösungen werden in der Reihenfolge aufgeführt, in der sie in „Programmieren lernen 1“ und „Programmieren lernen 2“ in der Swift Playgrounds App erscheinen, und nicht notwendigerweise in der Reihenfolge, in der sie in den Schüler- und Lehrerhandbüchern vorkommen.



Befehle

Befehle erteilen

Lösung 1

```
moveForward()  
moveForward()  
moveForward()  
collectGem()
```

Lösung 2

```
for i in 1..3 {  
    moveForward()  
}  
collectGem()
```

Einen neuen Befehl hinzufügen

Lösung 1

```
moveForward()  
moveForward()  
turnLeft()  
moveForward()  
moveForward()  
collectGem()
```

Lösung 2

```
for step in 1..2 {  
    moveForward()  
}  
turnLeft()  
for step in 1..2 {  
    moveForward()  
}  
collectGem()
```

Einen Schalter betätigen

Lösung 1

```
moveForward()  
moveForward()  
turnLeft()  
moveForward()  
collectGem()  
moveForward()  
turnLeft()  
moveForward()  
moveForward()  
toggleSwitch()
```

Lösung 2

```
for step in 1..2 {  
    moveForward()  
}  
turnLeft()  
moveForward()  
collectGem()  
moveForward()  
turnLeft()  
for step in 1..2 {  
    moveForward()  
}  
toggleSwitch()
```

Portalübung

Lösung 1

```
moveForward()  
moveForward()  
moveForward()  
turnLeft()  
moveForward()  
moveForward()  
toggleSwitch()  
moveForward()  
moveForward()  
turnLeft()  
moveForward()  
moveForward()  
collectGem()
```

Lösung 2

```
for step in 1..3 {  
    moveForward()  
}  
turnLeft()  
for step in 1..2 {  
    moveForward()  
}  
toggleSwitch()  
for step in 1..2 {  
    moveForward()  
}  
turnLeft()  
for step in 1..2 {  
    moveForward()  
}  
collectGem()
```

Befehle(Fortsetzung)

Bugs finden und beheben

Lösung 1

```
moveForward()
moveForward()
turnLeft()
moveForward()
collectGem()
moveForward()
toggleSwitch()
```

Übung zum Beheben von Bugs

Lösung 1

```
moveForward()
turnLeft()
moveForward()
moveForward()
toggleSwitch()
moveForward()
moveForward()
moveForward()
moveForward()
collectGem()
```

Lösung 2

```
moveForward()
turnLeft()
for step in 1..2 {
    moveForward()
}
toggleSwitch()
for step in 1..4 {
    moveForward()
}
collectGem()
```

Der kürzeste Weg

Lösung 1

```
moveForward()
moveForward()
moveForward()
collectGem()
moveForward()
moveForward()
moveForward()
moveForward()
toggleSwitch()
```

Lösung 2

```
for step in 1..3 {
    moveForward()
}
collectGem()
for step in 1..4 {
    moveForward()
}
toggleSwitch()
```

Funktionen

Ein neues Verhalten zusammenstellen

Lösung 1

```
moveForward()
moveForward()
moveForward()
turnLeft()
turnLeft()
turnLeft()
moveForward()
moveForward()
moveForward()
collectGem()
```

Lösung 2

```
for step in 1..3 {
    moveForward()
}
for step in 1..3 {
    turnLeft()
}
for step in 1..3 {
    moveForward()
}
collectGem()
```

Funktionen (Fortsetzung)

Eine neue Funktion erstellen

Lösung 1

```
func turnRight() {  
    turnLeft()  
    turnLeft()  
    turnLeft()  
}
```

```
moveForward()  
turnLeft()  
moveForward()  
turnRight()  
moveForward()  
turnRight()  
moveForward()  
turnRight()  
moveForward()  
turnLeft()  
moveForward()  
toggleSwitch()
```

Lösung 2

```
func turnRight() {  
    turnLeft()  
    turnLeft()  
    turnLeft()  
}
```

```
func goRight() {  
    moveForward()  
    turnRight()  
}
```

```
func goLeft() {  
    moveForward()  
    turnLeft()  
}
```

```
goLeft()  
goRight()  
goRight()  
goRight()  
goLeft()  
moveForward()  
toggleSwitch()
```

Einsammeln, Auslösen, Wiederholen

Lösung 1

```
func pickPlace() {  
    moveForward()  
    collectGem()  
    moveForward()  
    toggleSwitch()  
    moveForward()  
}
```

```
pickPlace()  
turnLeft()  
pickPlace()  
moveForward()  
turnLeft()  
pickPlace()  
turnLeft()  
pickPlace()
```

Funktionen (Fortsetzung)

Über das Brett

Lösung 1

```
func collectTwoGems() {
    collectGem()
    moveForward()
    collectGem()
    moveForward()
    turnRight()
}
```

```
collectTwoGems()
collectTwoGems()
collectTwoGems()
collectGem()
moveForward()
turnRight()
collectTwoGems()
```

Lösung 2

```
func collectTwoGems() {
    collectGem()
    moveForward()
    collectGem()
    moveForward()
    turnRight()
}
```

```
for i in 1..3 {
    collectTwoGems()
}
collectGem()
moveForward()
turnRight()
collectTwoGems()
```

Muster verschachteln

Lösung 1

```
func turnAround() {
    turnLeft()
    turnLeft()
}
```

```
func solveStair() {
    moveForward()
    collectGem()
    turnAround()
    moveForward()
    turnLeft()
}
```

```
solveStair()
solveStair()
solveStair()
solveStair()
```

Lösung 2

```
func turnAround() {
    turnRight()
    turnRight()
}
```

```
func solveStair() {
    moveForward()
    collectGem()
    turnAround()
    moveForward()
}
```

```
solveStair()
solveStair()
turnLeft()
solveStair()
solveStair()
```

Funktionen (Fortsetzung)

Tolle Treppen

Lösung 1

```
func collectGemTurnAround() {
    moveForward()
    moveForward()
    collectGem()
    turnLeft()
    turnLeft()
    moveForward()
    moveForward()
}
```

```
func solveRow() {
    collectGemTurnAround()
    collectGemTurnAround()
}
```

```
solveRow()
turnRight()
moveForward()
turnLeft()
solveRow()
turnRight()
moveForward()
turnLeft()
solveRow()
```

Schatzsuche

Lösung 1

```
func moveThenToggle() {
    moveForward()
    moveForward()
    toggleSwitch()
}
```

```
func toggleThenReturn() {
    moveThenToggle()
    turnLeft()
    turnLeft()
    moveForward()
    moveForward()
}
```

```
toggleThenReturn()
toggleThenReturn()
turnRight()
moveThenToggle()
toggleThenReturn()
moveForward()
moveForward()
moveThenToggle()
moveThenToggle()
```

for-Schleifen

Schleifen verwenden

Lösung 1

```
for i in 1..1 {
  moveForward()
  moveForward()
  collectGem()
  moveForward()
  moveForward()
  moveForward()
  collectGem()
  moveForward()
  moveForward()
  moveForward()
  collectGem()
  moveForward()
  moveForward()
  moveForward()
  collectGem()
  moveForward()
  moveForward()
  moveForward()
  collectGem()
  moveForward()
}
}
```

Lösung 2

```
for i in 1..5 {
  moveForward()
  moveForward()
  collectGem()
  moveForward()
}
}
```

Eine Schleife für alle Seiten

Lösung 1

```
for i in 1..4 {
  moveForward()
  collectGem()
  moveForward()
  moveForward()
  moveForward()
  turnRight()
}
}
```

Lösung 2

```
for i in 1..4 {
  moveForward()
  collectGem()
  for i in 1..3() {
    moveForward()
  }
  turnRight()
}
}
```

An den Rand und zurück

Lösung 1

```
for i in 1..4 {
  moveForward()
  moveForward()
  toggleSwitch()
  turnRight()
  turnRight()
  moveForward()
  moveForward()
  turnLeft()
}
}
```

for-Schleifen (Fortsetzung)

Schleifenspringer

Lösung 1

```
for i in 1..5 {
  moveForward()
  turnLeft()
  moveForward()
  moveForward()
  collectGem()
  turnRight()
}
```

Ausbreiten

Lösung 1

```
func traverseStairway() {
  for i in 1..7 {
    moveForward()
  }
}
```

```
func clearStairway() {
  traverseStairway()
  toggleSwitch()
  turnRight()
  turnRight()
  traverseStairway()
  turnRight()
}
```

```
for i in 1..3 {
  moveForward()
  moveForward()
  turnRight()
  clearStairway()
}
```


for-Schleifen (Fortsetzung)

Edelsteinfarm

Lösung 1

```
func turnAround() {
    turnLeft()
    turnLeft()
    moveForward()
    moveForward()
}

func solveRow() {
    turnRight()
    moveForward()
    collectGem()
    moveForward()
    collectGem()
    turnAround()
    moveForward()
    toggleSwitch()
    moveForward()
    toggleSwitch()
    turnAround()
    turnLeft()
    moveForward()
}

for i in 1...3 {
    solveRow()
}
```

Vier Lager leerräumen

Lösung 1

```
func turnAround() {
    turnRight()
    turnRight()
}

func collectFour() {
    collectGem()
    moveForward()
    collectGem()
    turnAround()
    moveForward()
    turnRight()
    moveForward()
    collectGem()
    turnAround()
    moveForward()
    moveForward()
    collectGem()
}

moveForward()

for i in 1...3 {
    collectFour()
    moveForward()
    moveForward()
}

collectFour()
```

Bedingte Anweisungen

Nach Schaltern suchen

Lösung 1

```
moveForward()
moveForward()

if isOnClosedSwitch {
    toggleSwitch()
}
moveForward()
if isOnClosedSwitch {
    toggleSwitch()
}
moveForward()
if isOnClosedSwitch {
    toggleSwitch()
}
```

Lösung 2

```
for i in 1..2 {
    moveForward()
}
for i in 1..2 {
    if isOnClosedSwitch {
        toggleSwitch()
    }
    moveForward()
}
if isOnClosedSwitch {
    toggleSwitch()
}
```

„else if“ verwenden

Lösung 1

```
moveForward()

if isOnClosedSwitch {}
    toggleSwitch()
} else if isOnGem {
    collectGem()
}

moveForward()
if isOnClosedSwitch {
    toggleSwitch()
} else if isOnGem {
    collectGem()
}
```

Schleifen mit bedingten Anweisungen

Lösung 1

```
for i in 1..12 {
    moveForward()
    if isOnClosedSwitch {
        toggleSwitch()
    } else if isOnGem {
        collectGem()
    }
}
```

Bedingte Anweisungen (Fortsetzung)

Bedingtes Klettern

Lösung 1

```
for i in 1...16 {
  if isOnGem {
    collectGem()
    turnLeft()
  } else {
    moveForward()
  }
}
```

Definieren intelligenter Funktionen

Lösung 1

```
func collectOrToggle() {
  moveForward()
  moveForward()
  if isOnGem {
    collectGem()
  } else if {
    isOnClosedSwitch {
      toggleSwitch()
    }
  }
}
collectOrToggle()
collectOrToggle()
turnLeft()
moveForward()
moveForward()
turnLeft()
collectOrToggle()
collectOrToggle()
turnRight()
moveForward()
turnRight()
collectOrToggle()
collectOrToggle()
```

Bedingte Anweisungen (Fortsetzung)

Eingesperrt

Lösung 1

```
func checkSquare() {
    if isOnGem {
        collectGem()
    } else if isOnClosedSwitch {
        toggleSwitch()
    }
}

func completeCorner() {
    checkSquare()
    moveForward()
    checkSquare()
    turnRight()
    moveForward()
}

moveForward()
turnRight()
for i in 1..4 {
    completeCorner()
}
```

Entscheidungsbaum

Lösung 1

```
func solveRightSide() {
    collectGem()
    turnRight()
    moveForward()
    moveForward()
    moveForward()
    turnLeft()
    moveForward()
    collectGem()
    turnLeft()
    turnLeft()
    moveForward()
    turnRight()
    moveForward()
    moveForward()
    moveForward()
    turnRight()
}

for i in 1..5 {
    moveForward()
    if isOnGem {
        solveRightSide()
    } else if isOnClosedSwitch {
        toggleSwitch()
        turnLeft()
        moveForward()
        collectGem()
        turnLeft()
        turnLeft()
        moveForward()
        turnLeft()
    }
}
```

Logische Operatoren

Den „NOT“-Operator verwenden

Lösung 1

```
for i in 1..4 {
  moveForward()
  if !isOnGem {
    turnLeft()
    moveForward()
    moveForward()
    collectGem()
    turnLeft()
    turnLeft()
    moveForward()
    moveForward()
    turnLeft()
  } else {
    collectGem()
  }
}
```

Die NOT-Spirale

Lösung 1

```
for i in 1..16 {
  if !isBlocked {
    moveForward()
  } else {
    turnLeft()
  }
}
toggleSwitch()
```

Dies UND das überprüfen

Lösung 1

```
for i in 1 ... 7 {
  moveForward()
  if isOnGem && isBlockedLeft {
    collectGem()
    turnRight()
    moveForward()
    moveForward()
    toggleSwitch()
    turnLeft()
    turnLeft()
    moveForward()
    moveForward()
    turnRight()
  } else if isOnGem {
    collectGem()
  }
}
```

Logische Operatoren (Fortsetzung)

Dies ODER das überprüfen

Lösung 1

```
for i in 1...12 {
  if isBlocked || isBlockedLeft {
    turnRight()
    moveForward()
  } else {
    moveForward()
  }
}
collectGem()
```

Logisches Labyrinth

Lösung 1

```
for i in 1 ... 8 {
  moveForward()
  if isOnGem && isOnClosedSwitch {
    turnRight()
    moveForward()
    moveForward()
    collectGem()
    turnLeft()
    turnLeft()
    moveForward()
    moveForward()
    turnRight()
    collectGem()
    toggleSwitch()
  } else if isOnClosedSwitch {
    turnLeft()
    toggleSwitch()
  }
  if isOnGem {
    collectGem()
  }
}
```

while-Schleifen

Code ausführen, solange ...

Lösung 1

```
while isOnClosedSwitch {
    toggleSwitch()
    moveForward()
}
```

Solution 2

```
while !isOnOpenSwitch {
    toggleSwitch()
    moveForward()
}
```

Intelligenter while-Schleifen erstellen

Lösung 1

```
while !isBlocked {
    if isOnClosedSwitch {
        toggleSwitch()
    }
    moveForward()
}
```

Das richtige Werkzeug auswählen

Lösung 1

```
func turnAndCollectGem() {
    moveForward()
    turnLeft()
    moveForward()
    collectGem()
    turnRight()
}

while !isBlocked {
    turnAndCollectGem()
}
```

Vier mal Vier

Lösung 1

```
for i in 1...4 {
    moveForward()
    moveForward()
    moveForward()
    if isOnClosedSwitch {
        toggleSwitch()
    }
    turnRight()
}
```

Solution 2

```
for i in 1...4 {
    for i in 1...3 {
        moveForward()
    }
    if isOnClosedSwitch {
        toggleSwitch()
    }
    turnRight()
}
```

while-Schleifen (Fortsetzung)

Umgedreht

Lösung 1

```
moveForward()
while isOnGem {
    turnLeft()
    collectGem()
    moveForward()
    collectGem()
    turnLeft()
    moveForward()
    turnRight()
    moveForward()
}
```

Schlaraffenland

Lösung 1

```
func solveColumn() {
    while !isBlocked {
        if isOnClosedSwitch {
            toggleSwitch()
        } else if isOnGem {
            collectGem()
        }
        moveForward()
    }
}
```

```
solveColumn()
turnRight()
moveForward()
turnRight()
solveColumn()
turnLeft()
moveForward()
turnLeft()
solveColumn()
```

Lösung 2

```
func solveColumn() {
    while !isBlocked {
        if isOnClosedSwitch {
            toggleSwitch()
        } else if isOnGem {
            collectGem()
        }
        moveForward()
    }
}
```

```
func solveAndRightTurn() {
    solveColumn()
    turnRight()
}
```

```
solveAndRightTurn()
moveForward()
solveAndRightTurn()
turnLeft()
moveForward()
turnLeft()
solveColumn()
```

Schleifen verschachteln

Lösung 1

```
while !isBlocked {
    while !isOnGem {
        moveForward()
    }
    collectGem()
    turnLeft()
}
```


while-Schleifen (Fortsetzung)

Zufällige Rechtecke

Lösung 1

```
while !isBlocked {
  while !isBlocked {
    moveForward()
  }
  turnRight()
}
toggleSwitch()
```

Du hast immer Recht

Lösung 1

```
while !isOnGem {
  while !isBlocked {
    moveForward()
    if isOnClosedSwitch {
      toggleSwitch()
    }
  }
  turnRight()
}

collectGem()
```

Algorithmen

Die Rechte-Hand-Regel

Lösung 1

```
func navigateAroundWall() {
  if isBlockedRight {
    moveForward()
  } else {
    turnRight()
    moveForward()
  }
}

while !isOnClosedSwitch {
  navigateAroundWall()
  if isOnGem {
    collectGem()
    turnLeft()
    turnLeft()
  }
}

toggleSwitch()
```

Algorithmen (Fortsetzung)

Deinen Algorithmus anpassen

Lösung 1

```
func navigateAroundWall() {
    if isBlockedRight && isBlocked {
        turnLeft()
    } else if isBlockedRight {
        moveForward()
    } else {
        turnRight()
        moveForward()
    }
}

while !isOnClosedSwitch {
    navigateAroundWall()
    if isOnGem {
        collectGem()
    }
}
toggleSwitch()
```

Ein Labyrinth durchqueren

Lösung 1

```
func navigateAroundWall() {
    if isBlockedRight && isBlocked {
        turnLeft()
    } else if isBlockedRight {
        moveForward()
    } else {
        turnRight()
        moveForward()
    }
}

while !isOnGem {
    navigateAroundWall()
}
collectGem()
```

In welche Richtung drehen?

Lösung 1

```
while !isOnGem {
    while !isOnClosedSwitch && !isOnGem {
        moveForward()
    }
    if isOnClosedSwitch && isBlocked {
        toggleSwitch()
        turnLeft()
    } else if isOnClosedSwitch {
        toggleSwitch()
        turnRight()
    }
}
collectGem()
```

Algorithmen (Fortsetzung)

Rolle nach rechts, Rolle nach links

Lösung 1

```
while !isOnOpenSwitch {
  moveForward()
  if isOnGem {
    collectGem()
    turnRight()
    moveForward()
    collectGem()
  } else if isOnClosedSwitch {
    toggleSwitch()
    turnLeft()
    moveForward()
    toggleSwitch()
  }
  while !isBlocked {
    moveForward()
  }
  if !isBlockedRight {
    turnRight()
  } else {
    turnLeft()
  }
}
```

Variablen

Den Überblick behalten

Lösung 1

```
var gemCounter = 0

moveForward()
moveForward()
collectGem()
gemCounter = 1
```

Den Wert hochtreiben

Lösung 1

```
var gemCounter = 0

moveForward()
collectGem()
gemCounter = 1
moveForward()
collectGem()
gemCounter = 2
moveForward()
collectGem()
gemCounter = 3
moveForward()
collectGem()
gemCounter = 4
moveForward()
collectGem()
gemCounter = 5
```

Solution 2

```
var gemCounter = 0

while gemCounter < 5 {
  moveForward()
  collectGem()
  gemCounter += 1
}
```

Variablen (Fortsetzung)

Einen Wert erhöhen

Lösung 1

```
var gemCounter = 0

while !isBlocked {
    while !isBlocked {
        if isOnGem {
            collectGem()
            gemCounter = gemCounter + 1
        }
        moveForward()
    }
    turnRight()
}
```

Auf der Suche nach sieben Edelsteinen

Lösung 1

```
var gemCounter = 0
while gemCounter < 7 {
    if isOnGem {
        collectGem()
        gemCounter = gemCounter + 1
    }
    if isBlocked {
        turnRight()
        turnRight()
    }
    moveForward()
}
```

Drei Edelsteine, vier Schalter

Lösung 1

```
var gemCounter = 0
var switchCounter = 0

while gemCounter != 3 || switchCounter != 4 {
    if gemCounter != 3 && isOnGem {
        collectGem()
        gemCounter = gemCounter + 1
    } else if switchCounter != 4 && isOnClosedSwitch {
        toggleSwitch()
        switchCounter = switchCounter + 1
    }
}

if isBlocked {
    turnRight()
    if isBlocked {
        turnLeft()
        turnLeft()
    }
}

moveForward()
}
```

Variablen (Fortsetzung)

Nach gleichen Werten suchen

Lösung 1

```
let switchCounter = numberOfSwitches

var gemCounter = 0

while gemCounter < switchCounter {
  if isOnGem {
    collectGem()
    gemCounter = gemCounter + 1
  }
  if isBlocked {
    turnRight()
  }
  moveForward()
}
```

Betätige die Schalter

Lösung 1

```
var gemCounter = 0
var switchCounter = 0

while !isOnClosedSwitch {
  while !isBlocked {
    if isOnGem {
      collectGem()
      gemCounter = gemCounter + 1
    }
    moveForward()
  }
  turnRight()
}

while switchCounter < gemCounter {
  while !isBlocked {
    if isOnClosedSwitch && switchCounter < gemCounter {
      toggleSwitch()
      switchCounter = switchCounter + 1
    }
    moveForward()
  }
  turnRight()
}
```

Variablen (Fortsetzung)

Gesamtwert berechnen

Lösung 1

```
let totalGems = randomNumberOfGems
var gemCounter = 0

while gemCounter < totalGems {
  if isOnGem {
    collectGem()
    gemCounter = gemCounter + 1
  }
  if isBlocked {
    turnRight()
    if isBlocked {
      turnLeft()
      turnLeft()
    }
    if isBlocked {
      turnLeft()
    }
  }
  moveForward()
}
```

Typen

Ein Portal deaktivieren

Lösung 1

```
greenPortal.isActive = false

func moveThree() {
  moveForward()
  moveForward()
  moveForward()
}

for i in 1...3 {
  moveThree()
  turnRight()
  moveThree()
  toggleSwitch()
  turnLeft()
  turnLeft()
}
```

Typen (Fortsetzung)

Portal Ein und Aus

Lösung 1

```
func moveAndCollect() {
    while !isBlocked {
        moveForward()
        if isOnGem {
            collectGem()
        }
    }
}

func turnAround() {
    turnLeft()
    turnLeft()
}

moveAndCollect()
turnAround()
purplePortal.isActive = false
while !isBlocked {
    moveForward()
}
toggleSwitch()
turnAround()
purplePortal.isActive = true
moveAndCollect()
```

Das richtige Portal aktivieren

Lösung 1

```
func moveCollect() {
    moveForward()
    collectGem()
}

func turnAround() {
    turnLeft()
    turnLeft()
}

moveForward()
moveCollect()
turnAround()
bluePortal.isActive = false
moveForward()
moveCollect()
turnAround()
bluePortal.isActive = true
pinkPortal.isActive = false
moveForward()
moveForward()
moveForward()
collectGem()
turnAround()
pinkPortal.isActive = true
moveForward()
turnAround()
moveCollect()
```

Typen (Fortsetzung)

Ecken der Welt

Lösung 1

```
func turnAround() {
    turnLeft()
    turnLeft()
}

func checkSquare() {
    if isOnGem {
        collectGem()
    } else if isOnClosedSwitch {
        toggleSwitch()
    }
}

func collectOrToggle() {
    moveForward()
    checkSquare()
    turnAround()
}

func collectOrToggleThenTurnRight() {
    collectOrToggle()
    moveForward()
    turnRight()
}

func collectOrToggleThenTurnLeft() {
    collectOrToggle()
    moveForward()
    turnLeft()
}

turnLeft()
moveForward()
moveForward()
greenPortal.isActive = false
for i in 1..3 {
    collectOrToggleThenTurnRight()
}
collectOrToggle()
greenPortal.isActive = true
moveForward()
greenPortal.isActive = false
collectOrToggleThenTurnLeft()
collectOrToggleThenTurnLeft()
moveForward()
moveForward()
orangePortal.isActive = false
moveForward()
for i in 1..3 {
    collectOrToggleThenTurnRight()
}
collectOrToggle()
orangePortal.isActive = true
moveForward()
orangePortal.isActive = false
turnLeft()
collectOrToggleThenTurnRight()
collectOrToggle()
```


Typen (Fortsetzung)

Edelsteine zufällig überall verteilen

Lösung 1

```
let totalGems = randomNumberOfGems
var gemCounter = 0

bluePortal.isActive = false
pinkPortal.isActive = false
while gemCounter < totalGems {
  if isOnGem {
    collectGem()
    gemCounter = gemCounter + 1
  }
  moveForward()
  if isBlocked {
    turnLeft()
    turnLeft()
    if bluePortal.isActive == true {
      bluePortal.isActive = false
      pinkPortal.isActive = false
    } else if bluePortal.isActive ==
false {
      bluePortal.isActive = true
      pinkPortal.isActive = true
    }
  }
}
```

Initialisierung

Deinen Experten initialisieren

Lösung 1

```
let expert = Expert()

func solveSide() {
  expert.moveForward()
  expert.moveForward()
  expert.moveForward()
  if expert.isOnGem {
    expert.collectGem()
  } else {
    expert.turnLockUp()
  }
}

func returnToCenter() {
  expert.turnLeft()
  expert.turnLeft()
  expert.moveForward()
  expert.moveForward()
  expert.moveForward()
  expert.turnRight()
}

for i in 1...3 {
  solveSide()
  returnToCenter()
}
solveSide()
```

Initialisierung (Fortsetzung)

Deinen Experten trainieren

Lösung 1

```
let expert = Expert()

func turnAround() {
    expert.turnLeft()
    expert.turnLeft()
    expert.moveForward()
    expert.moveForward()
}

func completeSide() {
    expert.moveForward()
    expert.moveForward()
    expert.collectGem()
}

for i in 1 ... 2 {
    completeSide()
    turnAround()
    expert.turnRight()
}
completeSide()
expert.turnLockDown()
turnAround()
expert.turnRight()

for i in 1 ... 3 {
    expert.moveForward()
}

expert.turnLeft()
for i in 1 ... 3 {
    completeSide()
    turnAround()
    expert.turnLeft()
}
```

Instanzen verschiedener Typen verwenden

Lösung 1

```
let expert = Expert()
let character = Character()

expert.moveForward()
expert.turnLockUp()
character.moveForward()
character.collectGem()
character.moveForward()
character.turnRight()
character.moveForward()
character.moveForward()
expert.turnLockDown()
expert.turnLockDown()
character.moveForward()
character.collectGem()
```

Initialisierung (Fortsetzung)

Dazu gehören immer zwei

Lösung 1

```
let expert = Expert()
let character = Character()

func turnCorner() {
    expert.moveForward()
    expert.moveForward()
    expert.turnRight()
    expert.moveForward()
    expert.moveForward()
}
expert.turnLeft()
expert.moveForward()
turnCorner()
expert.turnLeft()
expert.turnLockDown()
expert.turnLockDown()
character.moveForward()
character.moveForward()
character.collectGem()
expert.turnRight()
turnCorner()
expert.moveForward()
expert.moveForward()
turnCorner()
expert.turnLeft()
expert.turnLockUp()
character.moveForward()
character.moveForward()
character.toggleSwitch()
```

Parameter

Weiter vorwärts bewegen

Lösung 1

```
let expert = Expert()

func move(distance: Int) {
    for i in 1..distance {
        expert.moveForward()
    }
}

move(distance: 6)
expert.turnRight()
expert.move(distance: 2)
expert.turnRight()
move(distance: 5)
expert.turnLeft()
move(distance: 5)
expert.turnLeft()
expert.turnLockUp()
expert.turnLeft()
move(distance: 3)
expert.turnRight()
move(distance: 3)
expert.turnRight()
move(distance: 4)
expert.collectGem()
```

Lösung 2

```
let expert = Expert()
func move(distance: Int) {
    for i in 1..distance {
        expert.moveForward()
    }
}
func solvePuzzle(factorOrAddened: Int) {
    expert.move(distance: factorOrAddend * 3)
    expert.turnRight()
    expert.move(distance: factorOrAddend)
    expert.turnRight()
    for i in 1..2 {
        expert.move(distance:
factorOrAddend + 3)
        expert.turnLeft()
    }
    expert.turnLockUp()
    expert.turnLeft()
    for i in 1..2 {
        expert.move(distance:
factorOrAddend + 1)
        expert.turnRight()
    }
    expert.move(distance: factorOrAddend
* 2)
    expert.collectGem()
}
solvePuzzle(factorOrAddend: 2)
```

Parameter (Fortsetzung)

Nach oben und unten kurbeln

Lösung 1

```
let expert = Expert()
let character = Character()

func turnAround() {
    character.turnLeft()
    character.turnLeft()
}

func collectGemTurnAround() {
    character.moveForward()
    character.moveForward()
    character.collectGem()
    turnAround()
    character.moveForward()
    character.moveForward()
    character.turnRight()
}

for i in 1...4 {
    expert.turnLock(up: true,
numberOfTimes: 4)
    expert.turnRight()
}
for i in 1...3 {
    while !character.isOnGem {
        character.moveForward()
    }
    character.collectGem()
    character.turnRight()
}
character.moveForward()
for i in 1...4 {
    expert.turnLock(up: false,
numberOfTimes: 3)
    expert.turnRight()
}
character.turnLeft()
character.moveForward()
character.collectGem()
turnAround()
for i in 1...3 {
    character.moveForward()
    character.moveForward()
    if !character.isOnGem {
        character.turnRight()
        collectGemTurnAround()
    } else {
        character.collectGem()
    }
}
}
```

Parameter (Fortsetzung)

An einem bestimmten Ort platzieren

Lösung 1

```
let expert = Expert()
world.place(expert, atColumn: 1, row: 1)
expert.collectGem()
world.place(expert, atColumn: 1, row: 6)
expert.collectGem()
world.place(expert, atColumn: 6, row: 1)
expert.collectGem()
```

Lösung 2

```
let expert = Expert()
world.place(expert, atColumn: 2, row: 6)

func turnAround() {
    expert.turnLeft()
    expert.turnLeft()
}

func turnLockCollectGem() {
    expert.turnLeft()
    expert.turnLockUp()
    turnAround()
    expert.moveForward()
    expert.collectGem()
    turnAround()
    expert.moveForward()
    expert.turnRight()
}

turnLockCollectGem()
expert.move(distance: 5)
turnLockCollectGem()
expert.move(distance: 6)
expert.collectGem()
```

Flüsse überqueren

Lösung 1

```
let expert = Expert()
world.place(expert, facing: .south,
atColumn: 1, row: 8)

func collectGemsInLine() {
    while !expert.isBlocked {
        if expert.isOnGem {
            expert.collectGem()
        }
        expert.moveForward()
    }
}

collectGemsInLine()
expert.turnLockDown()
expert.turnLeft()
collectGemsInLine()
expert.turnLockUp()
expert.turnRight()
collectGemsInLine()
```

Parameter (Fortsetzung)

Zwei Charaktere platzieren

Lösung 1

```
let character = Character()
let expert = Expert()

world.place(character, facing: north, atColumn: 0, row: 0)
world.place(expert, facing: north, atColumn: 3, row: 0)

func collectAndJump() {
    for i in 1 ... 2 {
        character.collectGem()
        character.jump()
        character.jump()
    }
}

expert.toggleSwitch()
expert.turnLockUp()

collectAndJump()
character.turnRight()
collectAndJump()
character.turnLeft()
character.collectGem()
character.move(distance: 2)
character.collectGem()
```

Zwei Experten

Lösung 1

```
let topExpert = Expert()
let bottomExpert = Expert()

world.place(topExpert, facing: north, atColumn: 0, row: 4)
world.place(bottomExpert, facing: east, atColumn: 0, row: 0)

bottomExpert.collectGem()
bottomExpert.move(distance: 3)
bottomExpert.turnLeft()
bottomExpert.turnLock(up: true, numberOfTimes: 2)
bottomExpert.turnRight()
topExpert.turnLockDown()
bottomExpert.move(distance: 3)
bottomExpert.turnLock(up: false, numberOfTimes: 2)
topExpert.turnRight()

while !topExpert.isBlocked {
    if topExpert.isOnGem {
        topExpert.collectGem()
    }
    topExpert.moveForward()
}
```

Parameter (Fortsetzung)

Twin Peaks

Lösung 1

```
let totalGems = randomNumberOfGems

let expert = Expert()
let character = Character()
world.place(expert, facing: north, atColumn: 0, row: 4)
world.place(character, facing: north, atColumn: 2, row: 0)

var gemCounter = 0
var platformPosition = 0

func jumpAcrossSide() {
  for i in 1..6 {
    if character.isOnGem && gemCounter < totalGems {
      character.collectGem()
      gemCounter = gemCounter + 1
    }
    character.jump()
  }
}

while gemCounter < totalGems {
  jumpAcrossSide()
  character.turnRight()
  if platformPosition == 0 {
    expert.turnLockUp()
    platformPosition = 1
  } else if platformPosition == 3 {
    expert.turnLock(up: false, numberOfTimes: 2)
    platformPosition = 1
  }
  character.moveForward()
  if character.isOnGem && gemCounter < totalGems {
    character.collectGem()
    gemCounter = gemCounter + 1
  }
  if platformPosition == 1 {
    expert.turnLock(up: true, numberOfTimes: 2)
    platformPosition = 3
  }
  character.moveForward()
  character.turnRight()
}
```


Weltenbau

Welten verbinden

Lösung 1

```
let block1 = Block()
world.place(block1, atColumn: 3, row: 3)

while !isOnClosedSwitch {
    moveForward()
    if isBlocked {
        turnLeft()
        if isBlocked {
            turnRight()
            turnRight()
        }
    }
}
toggleSwitch()
```

Verbinden und Lösen

Lösung 1

```
let block1 = Block()
let block2 = Block()
let block3 = Block()
let block4 = Block()
let block5 = Block()

world.place(block1, atColumn: 2, row: 2)
world.place(block2, atColumn: 2, row: 2)

world.place(block3, atColumn: 4, row: 2)

world.place(block4, atColumn: 6, row: 2)
world.place(block5, atColumn: 6, row: 2)

func crossBridge() {
    turnRight()
    move(distance: 4)
    collectGem()
    turnLeft()
    turnLeft()
    move(distance: 4)
    turnRight()
}

for i in 1..3 {
    move(distance: 2)
    toggleSwitch()
    crossBridge()
}
```

Weltenbau (Fortsetzung)

Mach deine eigenen Portale

Lösung 1

```
let greenPortal = Portal(color: .green)
world.place(greenPortal, atStartColumn: 1, startRow: 5, atEndColumn: 5, endRow: 1)

var gemCounter = 0
while gemCounter < 8 {
    moveForward()
    if gemCounter == 4 {
        turnLeft()
        turnLeft()
    } else {
        turnLeft()
    }
    moveForward()
    collectGem()
    gemCounter = gemCounter + 1
    turnLeft()
    turnLeft()
}
```

Weltenbau (Fortsetzung)

Die Treppe hinauf

Lösung 1

```
world.place(Stair(), facing: south, atColumn: 3, row: 1)
world.place(Stair(), facing: south, atColumn: 3, row: 3)
world.place(Stair(), facing: west, atColumn: 1, row: 4)
world.place(Stair(), facing: west, atColumn: 1, row: 6)
world.place(Stair(), facing: east, atColumn: 5, row: 6)
world.place(Stair(), facing: north, atColumn: 2, row: 7)
world.place(Stair(), facing: north, atColumn: 4, row: 7)
```

```
func toggleSide() {
    toggleSwitch()
    while !isBlocked {
        moveForward()
        toggleSwitch()
    }
}
```

```
func turnCorner() {
    turnRight()
    move(distance: 2)
    turnLeft()
    move(distance: 2)
    turnRight()
}
```

```
move(distance: 4)
turnLeft()
move(distance: 3)
turnRight()
for i in 1 ... 2 {
    toggleSide()
    turnCorner()
}
toggleSide()
```

Weltenbau (Fortsetzung)

Schwebende Inseln

Lösung 1

```
let character = Character()
world.place(character, facing: south, atColumn: 1, row: 7)

func completeIsland() {
    character.toggleSwitch()
    character.jump()
    character.collectGem()
    character.turnLeft()
    character.jump()
    character.toggleSwitch()
}

completeIsland()
world.place(character, facing: north, atColumn: 6, row: 3)
completeIsland()
world.place(character, facing: east, atColumn: 1, row: 1)
completeIsland()
```

Eine Schleife bauen

Lösung 1

```
let totalGems = randomNumberOfGems
var gemCounter = 0

world.place(Block(), atColumn: 0, row: 2)
world.place(Block(), atColumn: 3, row: 3)

let expert = Expert()
world.place(expert, facing: east, atColumn: 2, row: 3)

while gemCounter < totalGems {
    if expert.isOnGem {
        expert.collectGem()
        gemCounter = gemCounter + 1
    }
    if expert.isBlocked {
        expert.turnRight()
        if expert.isBlocked {
            expert.turnRight()
            if expert.isBlocked {
                expert.turnRight()
            }
        }
    }
    expert.moveForward()
}
```

Weltenbau (Fortsetzung)

Dein eigenes Rätsel

Lösung 1

```
world.place(Gem(), atColumn:2, row: 3)
world.place(Switch(), atColumn: 2, row: 4)
world.removeItem(atColumn: 2, row: 3)
world.removeItem(atColumn: 3, row: 4)
```

Arrays

Informationen speichern

Lösung 1

```
var rows = [0,1,2,3,4,5]
placeCharacters(at: rows)
```

Iterationsexploration

Lösung 1

```
let columns = [0, 1, 2, 3, 4]
for currentColumn in columns {
    world.place(Gem(), atColumn: currentColumn, row: 1)
    world.place(Switch(), atColumn: currentColumn, row: 1)
}
```

Blöcke stapeln

Lösung 1

```
let blockLocations = [
    Coordinate(column: 0, row: 0),
    Coordinate(column: 3, row: 3),
    Coordinate(column: 0, row: 3),
    Coordinate(column: 3, row: 0)
]
for coordinate in blockLocations {
    for i in 1 ... 5 {
        world.place(Block(), at: coordinate)
    }
}
```

Arrays (Fortsetzung)

In die richtige Reihenfolge bringen

Lösung 1

```
var characters: [Item] = [
    Character(name: .blu),
    Portal(color: .pink),
    Character(name: .byte),
    Gem(),
    Character(name: .hopper)
]
// Remove the portal
characters.remove(at: 1)
// Remove the gem
characters.remove(at:2)
// Insert the Expert behind Byte
characters.insert(Expert(), at: 1)

var rowPlacement = 0
for character in characters {
    world.place(character, at: Coordinate(column: 1, row: rowPlacement))
    rowPlacement += 1
}
```

Zu einem Array hinzufügen

Lösung 1

```
let allCoordinates = world.allPossibleCoordinates
var blockSet: [Coordinate] = []

for coordinate in allCoordinates {
    if coordinate.column > 5 || coordinate.row < 4 {
        blockSet.append(coordinate)
    }
}
for coordinate in blockSet {
    for i in 1..6 {
        world.place(Block(), at: coordinate)
    }
}
}
```

Arrays (Fortsetzung)

Inselbaumeister

Lösung 1

```
let allCoordinates = world.allPossibleCoordinates
// Create two empty arrays of type [Coordinate].
var island: [Coordinate] = []
var sea: [Coordinate] = []

for coordinate in allCoordinates {
    if coordinate.column >= 3 && coordinate.column < 7 && coordinate.row > 3 && coordinate.row < 7 {
        island.append(coordinate)
    } else {
        sea.append(coordinate)
    }
}

// For your island, array, place blocks.
for coordinate in island {
    for i in 1..4 {
        world.place(Block(), at: coordinate)
    }
}

// For your sea, array, place water.
for coordinate in sea {
    world.removeAllBlocks(at: coordinate)
    world.place(Water(), at: coordinate)
}
```

Entfernte Werte hinzufügen

Lösung 1

```
// Create an array of all coordinates in row 2
var row2 = world.coordinates(inRows: [2])

// Create an empty array of coordinates
var discardedCoordinates: [Coordinate] = []

for i in 1..12 {
    for coordinate in row2 {
        world.place(Block(), at: coordinate)
    }
    // Remove a coordinate and append it to your empty array
    discardedCoordinates.append(row2.remove(at: 0))
}

// Place a character for each coordinate added to your empty array.
for coordinate in discardedCoordinates {
    world.place(Character(), at: coordinate)
}
```

Arrays (Fortsetzung)

Arraygrenzenfehler beheben

Lösung 1

```
var teamBlu: [Character] = []

// note how many characters are in your array
for i in 1..9 {
    teamBlu.append(Character(name: .blu))
}

var columnPlacement = 0
for blu in teamBlu {
    world.place(blu, at: Coordinate(column: columnPlacement, row: 4))
    columnPlacement += 1
}

// find the array out of bounds error
teamBlu[0].jump()
teamBlu[2].collectGem()
teamBlu[4].jump()
teamBlu[6].collectGem()
teamBlu[8].jump()
```

Eine Landschaft erzeugen

Lösung 1

```
var heights: [Int] = [1,0,8,9,4,3,1,6,12,5]
let allCoordinates = world.allPossibleCoordinates

var index = 0
for coordinate in allCoordinates {
    if index == heights.count {
        index = 0
    }
    for i in 0..heights[index] {
        world.place(Block(), at: coordinate)
    }
    index += 1
}
```

Arrays (Fortsetzung)

Zufällig erzeugtes Land

Lösung 1

```
let allCoordinates = world.allPossibleCoordinates
var heights: [Int] = []

// Append random numbers to heights.
for i in 1..12 {
    heights.append(randomInt(from: 0, to: 8))
}

var index = 0
for coordinate in allCoordinates {
    if index == heights.count {
        index = 0
    }

    // currentHeight stores the height at the current index.
    var currentHeight = heights[index]

    if currentHeight == 0 {
        // Do something interesting if currentHeight is equal to 0.
        world.removeItem(at: coordinate)
    } else {
        for i in 1..currentHeight {
            world.place(Block(), at: coordinate)
        }
        if currentHeight > 5 {
            // Do something different, such as placing a character.
            world.place(Character(), at: coordinate)
        } else if coordinate.column >= 3 && coordinate.column < 6 {
            // Do something different, such as placing water.
            world.removeItem(at: coordinate)
            world.place(Water(), at: coordinate)
        }

        // Add more rules to customize your world.
    }

    index += 1
}
}
```

Arrays (Fortsetzung)

Eine andere Methode, ein Array zu erstellen

Lösung 1

```
let allCoordinates = world.allPossibleCoordinates

for coordinate in allCoordinates {
  let height = coordinate.column + coordinate.row

  for i in 0...height {
    world.place(Block(), at: coordinate)
  }

  if height >= 8 && height < 10 {
    world.place(Character(name: .blu), at: coordinate)
  } else if height > 9 {
    world.place(Character(name: .byte), at: coordinate)
  }
}

let characters = world.existingCharacters(at: allCoordinates)

for character in characters {
  character.jump()
}
```

Arrays (Fortsetzung)

Die Kunst des Arrays

Lösung 1

```
// Create coordinate zones.
let allCoordinates = world.allPossibleCoordinates
let backRow = world.coordinates(inRows: [9])
let insideSquare = world.coordinates(inColumns: [4,5], intersectingRows: [4,5])
let squareCorners = world.coordinates(inColumns: [2,3,6,7], intersectingRows: [3,7])

// Place platform locks.
let squareLock = PlatformLock(color: .green)
world.place(squareLock, at: Coordinate(column: 1, row: 1))
let cornerLock = PlatformLock(color: .pink)
world.place(cornerLock, at: Coordinate(column: 8, row: 1))
let backLock = PlatformLock(color: .blue)
world.place(backLock, at: Coordinate(column: 4, row: 1))

// Place characters and platforms.
for coor in insideSquare {
    world.place(Platform(onLevel: 4, controlledBy: squareLock), at: coor)
    world.place(Character(name: .hopper), at: coor)
}

for coor in squareCorners {
    world.place(Platform(onLevel: 4, controlledBy: cornerLock), at: coor)
    world.place(Expert(), at: coor)
}

for coor in backRow {
    world.place(Platform(onLevel: 2, controlledBy: backLock), at: Coordinate(column:
coor.column, row: coor.row + 1))
    world.place(Character(name: .blu), facing: north, at: coor)
}

// Create arrays from existing characters.
let blus = world.existingCharacters(at: backRow)
let hoppers = world.existingCharacters(at: insideSquare)
let experts = world.existingExperts(at: squareCorners)

// Do cool stuff.
squareLock.movePlatforms(up: true, numberOfTimes: 3)

for hopper in hoppers {
    hopper.turnUp()
}

cornerLock.movePlatforms(up: true, numberOfTimes: 7)

for expert in experts {
    expert.collectGem()
}

for blu in blus {
    blu.jump()
}
backLock.movePlatforms(up: true, numberOfTimes: 11)

for blu in blus {
    blu.jump()
}
```

Arrays (Fortsetzung)

Weltenbau

Lösung 1

```
for coordinate in world.row(7) {  
world.place(Character(name: .blu), at: coordinate)  
}
```

```
for coordinate in world.row(5) {  
world.place(Character(name: .byte), at: coordinate)  
}
```

```
for coordinate in world.row(3) {  
world.place(Character(name: .hopper), at: coordinate)  
}
```
