Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 01

TECHNISCHE UNIVERSITÄT DARMSTADT

Entwurf

Achtung: Dieses Dokument ist ein Entwurf und ist noch nicht zur Bearbeitung/Abgabe freigegeben. Es kann zu Änderungen kommen, die für die Abgabe relevant sind. Es ist möglich, dass sich **alle** Aufgaben noch grundlegend ändern. Es gibt keine Garantie, dass die Aufgaben auch in der endgültigen Version überhaupt noch vorkommen und es wird keine Rücksicht auf bereits abgegebene Lösungen genommen, die nicht die Vorgaben der endgültigen Version erfüllen.

Vorbereitende Übungen

Hinweis (Für die Vorübungen)**:**

Alle Vorübungen können Sie mit Hilfe unser FopBot-Playground-Vorlage bearbeiten. Darüber hinaus werden keine Klassen oder Methoden bereitgestellt, die Sie nicht selbst implementieren müssen. Wenn nicht explizit angegeben, wird die Standardgröße der Welt verwendet (10x10 Felder).

V1: Theorie

V1.1: Was war das nochmal für ein Operator?

★☆☆☆☆

Legen Sie eine Variable `int a` an und setzen Sie ihren Wert auf 127. Jetzt legen Sie eine weitere Variable int b an und setzen Sie ihren Wert auf 42. Was gibt nun der Ausdruck `int c = a % b;` wieder? Beschreiben Sie in Ihren eigenen Worten, welche Berechnung mit dem %-Operator durchgeführt wird.

Lösungsvorschlag:

Der Ausdruck gibt 1 zurück, also den Rest der ganzzahligen Division von a und b. Beim % handelt sich um den Modulo-Operator, welcher den Rest einer ganzzahligen Division zurückgibt.

V1.2: Bedingungen I

★☆☆☆☆

Betrachten Sie folgenden Codeausschnitt:

```
Robot bot1 = new Robot(3,1,UP,1);
bot1.move();
if(bot1.isNextToACoin()) {
    bot1.pickCoin();
}
else {
    bot1.putCoin();
}
```

Beschreiben Sie in eigenen Worten, welchem Zweck dieser Codeausschnitt dient. Erweitern Sie außerdem den Code so, dass `bot1` nur einen Coin ablegt, wenn er auch mindestens einen besitzt.

Lösungsvorschlag:

Es wird ein Roboter an der Position `(3, 1)` mit der Blickrichtung nach oben platziert. Dieser Roboter hat eine Münze und führt einen Schritt nach vorne (in Blickrichtung) aus. Falls sich unter ihm eine Münze befindet, so hebt er diese auf. Ansonsten legt er eine Münze ab (Erweiterung: Sofern er mindestens eine Münze besitzt).

```
Robot bot1 = new Robot(3, 1, UP, 1);
bot1.move();
if (bot1.isNextToACoin()) {
    bot1.pickCoin();
} else if (bot1.hasAnyCoins()) {
    bot1.putCoin();
}
```

V1.3: Bedingungen II

Ihr Kommilitone ist etwas tippfaul und lässt deswegen gerne einmal Klammern weg, um sich Arbeit zu sparen. Er hat in seinem Code eine Variable `int number` angelegt, in der er eine Zahl speichert. Ist diese Zahl kleiner als 0, so möchte er das Vorzeichen der Zahl umdrehen und sie anschließend um 1 erhöhen. Ist die Zahl hingegen größer als 0, so möchte er die Zahl verdoppeln. Dazu schreibt er folgenden Code:

```
if(number < 0) number = -number;
number = number + 1;
else number = number * 2;
```

Kann der Code so ausgeführt werden? Beschreiben Sie den Fehler, den ihr Kommilitone begangen hat.

Nachdem Sie ihren Kommilitonen auf den obigen Fehler hingewiesen haben, überarbeitet er seinen Versuch. Wie sieht es mit folgender Variante aus?

```
if(counter > 0) counter = counter * 2;
if(counter < 0) counter = -counter;
counter = counter + 1;
```

Falls der Code auch nicht funktioniert, wie würde der korrekte Code aussehen?

Lösungsvorschlag:

1. Zwischen einer `if`- und `else`-Anweisung darf nur eine Anweisung oder ein Anweisungsblock stehen. Der Komilitone müsste die erste und zweite Anweisung nach dem `if` in Klammern schreiben.

2. Hier fehlen die Klammern bei der zweiten Anweisung nach dem `if`, da der `counter` ansonsten immer um 1 inkrementiert wird.

   Der verbesserte Code lautet:

```
if (counter > 0)
    counter = counter * 2;
if (counter < 0) {
    counter = -counter;
    counter = counter + 1;
}
```

V2: Praxis

V2.1: Wechselgeld ★★★☆☆

Sie wollen einen Münzautomaten programmieren, der Wechselgeld ausgibt. Dieser hat Münzen im Wert von 1ct, 2ct, 5ct, 10ct, 20ct, 50ct, 1€ und 2€. Um die Mechanik des Automaten zu schonen, soll der Automat immer die minimale Anzahl an Münzen ausgeben, um den Betrag zu erreichen. Legen Sie zunächst eine Variable `change` vom Typ `double` an und initialisieren Sie diese mit einem beliebigen positiven Wert. Dieser Wert soll den Betrag in Euro darstellen, für den Sie Wechselgeld ausgeben wollen. Schreiben Sie ein Programm, das die minimale Anzahl an Münzen berechnet. Außerdem soll ausgegeben werden, wie viele Münzen von jedem Wert verwendet wurden. Die Ausgabe könnte beispielsweise so aussehen:

Ausgabe

```
Gesamtbetrag: 1234.56€
======================
2€  : 617
1€  : 0
50ct: 1
20ct: 0
10ct: 1
5ct : 1
2ct : 0
1ct : 1
======================
Münzen insgesamt: 621
```

Verbindliche Anforderung:

Implementieren Sie die Ausgabe des Wechselgelds mit Hilfe einer einzigen While-Schleife. Innerhalb der While-Schleife ziehen Sie immer den größtmöglichen Münzwert vom Gesamtbetrag ab und erhöhen die Anzahl der Münzen dieses Wertes um eins. Wenn der Gesamtbetrag 0 ist, beenden Sie die Schleife. Hier dürfen Sie noch kein Modulo verwenden.

Testen Sie ihre Ausgabe mit den folgenden Geldbeträgen für `change`:

- 2€
- 5€
- 1234.56€
- 9999.99€

Lösungsvorschlag:

- 2€: 1x 2€, 0x 1€, 0x 50ct, 0x 20ct, 0x 10ct, 0x 5ct, 0x 2ct, 0x 1ct; Gesamt: 1 Münze
- 5€: 2x 2€, 1x 1€, 0x 50ct, 0x 20ct, 0x 10ct, 0x 5ct, 0x 2ct, 0x 1ct; Gesamt: 3 Münzen
- 1234.56€: 617x 2€, 0x 1€, 1x 50ct, 0x 20ct, 0x 10ct, 1x 5ct, 0x 2ct, 1x 1ct; Gesamt: 620 Münzen

- 9999.99€: 4999x 2€, 1x 1€, 1x 50ct, 2x 20ct, 0x 10ct, 1x 5ct, 2x 2ct, 0x 1ct; Gesamt: 5006 Münzen

Wechselgeld

```java
double change = 1234.56;

System.out.println("Gesamtbetrag: " + change + "€");
System.out.println("==================================");

int twoEuro = 0, oneEuro = 0, fiftyCent = 0, twentyCent = 0, tenCent = 0,
    fiveCent = 0, twoCent = 0, oneCent = 0;

while (change >= 0) {
    if (change >= 2) {
        twoEuro++;
        change -= 2;
    } else if (change >= 1) {
        oneEuro++;
        change -= 1;
    } else if (change >= 0.5) {
        fiftyCent++;
        change -= 0.5;
    } else if (change >= 0.2) {
        twentyCent++;
        change -= 0.2;
    } else if (change >= 0.1) {
        tenCent++;
        change -= 0.1;
    } else if (change >= 0.05) {
        fiveCent++;
        change -= 0.05;
    } else if (change >= 0.02) {
        twoCent++;
        change -= 0.02;
    } else {
        oneCent++;
        change -= 0.01;
    }
}

System.out.println("2€  : " + twoEuro);
System.out.println("1€  : " + oneEuro);
System.out.println("50ct: " + fiftyCent);
System.out.println("20ct: " + twentyCent);
System.out.println("10ct: " + tenCent);
System.out.println("5ct : " + fiveCent);
System.out.println("2ct : " + twoCent);
System.out.println("1ct : " + oneCent);
System.out.println("==================================");
System.out.println("Münzen gesamt: " + (twoEuro + oneEuro + fiftyCent +
    twentyCent + tenCent + fiveCent + twoCent + oneCent));
```

Anmerkung (Nicht Klausurrelevant)**:**

Einen solchen Algorithmus, der lokal immer die Entscheidung trifft, die den größten Gewinn bringt, nennt man auch „Greedy-Algorithmus“.[1]

V2.2: Wechselgeld in Schlau

★★★½☆

Lösen Sie jetzt die gleiche Aufgabe, aber nutzen Sie den Modulo-Operator (%) und die Division (/). Sie können die Ausgabe aus der vorherigen Aufgabe verwenden, um die Anzahl der Münzen zu überprüfen.

Hinweis:

Wandeln Sie den Betrag zunächst in Cent um.

Lösungsvorschlag:

Wechselgeld in Schlau

```java
double change = 1234.56;

System.out.println("Gesamtbetrag: " + change + "€");
System.out.println("==================================");

int changeInCents = (int) (change * 100); // alternativ mit float
int totalCoins = 0;
for (int i = 200; i >= 1; i /= 2) {
    if (i==25) { // nach 50 cent kommt 20 cent, nicht 25
        i=20;
    }
    int count = changeInCents / i;
    changeInCents %= i;
    totalCoins += count;
    System.out.println(i + "cent(s): " + count);
}

System.out.println("==================================");
System.out.println("Münzen gesamt: " + (totalCoins));
```

V3: Challenge - Der Navigator

★★★★☆

Gegeben seinen die folgenden fünf Variablen:

- `int startX`: Die x-Koordinate des Startpunktes
- `int startY`: Die y-Koordinate des Startpunktes
- `Direction startDirection`: Die Blickrichtung des Roboters am Startpunkt
- `int destinationX`: Die x-Koordinate des Zielpunktes

[1] Siehe `https://de.wikipedia.org/wiki/Greedy-Algorithmus`. In der AuD werden Sie diesen Algorithmus noch genauer kennenlernen.

- `int destinationY`: Die y-Koordinate des Zielpunktes

Ihr Roboter befindet sich zu Beginn an der Position (`startX`, `startY`) und schaut in die gegebene `startDirection`. Schreiben Sie ein Programm, das den Roboter vom Startpunkt zum Zielpunkt navigiert. Sie können davon ausgehen, dass es keine Hindernisse in der Welt gibt und die Koordinaten des Zielpunktes immer erreichbar sind.

Hinweise:

- Es empfiehlt sich, Hilfsmethoden zu schreiben, die den Roboter in eine bestimmte Richtung drehen lassen.
- Auch wenn die Aufgabe recht kompliziert scheint, lässt sich in unter 40 Zeilen lösen. (das ist keine Anforderung, aber ein Hinweis, dass Sie nicht zu kompliziert denken sollten)

Lösungsvorschlag:

Der Navigator

```java
public static void main(String[] args) {
    World.setSize(10, 10);
    World.setDelay(100);
    World.setVisible(true);

    int startX = 1;
    int startY = 1;
    Direction startDir = Direction.RIGHT;
    int destinationX = 5;
    int destinationY = 5;

    Robot r = new Robot(startX, startY, startDir, 0);
    while (r.getX() != destinationX || r.getY() != destinationY) {
        if (r.getX() < destinationX) {
            turnTo(r, Direction.RIGHT);
        } else if (r.getX() > destinationX) {
            turnTo(r, Direction.LEFT);
        } else if (r.getY() < destinationY) {
            turnTo(r, Direction.DOWN);
        } else {
            turnTo(r, Direction.UP);
        }
        r.move();
    }
}

public void turnTo(Robot r, Direction d) {
    while (r.getDirection() != d) {
        r.turnLeft();
    }
}
```

Hausübung 01

Gesamt: 16 Punkte

Foreign Contaminants

Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben* im Moodle-Kurs.

Verstöße gegen verbindliche Anforderungen führen zu Punktabzügen und können die korrekte Bewertung Ihrer Abgabe beeinflussen. Sofern vorhanden, müssen die in der Vorlage mit `TODO` markierten `crash`-Aufrufe entfernt werden. Andernfalls wird die jeweilige Aufgabe nicht bewertet.

Die für diese Hausübung relevanten Verzeichnisse sind `src/main/java/h01` und ggf. `src/test/java/h01`.

Verbindliche Anforderungen für die gesamte Hausübung:

- Alle Zufallszahlen werden mit der Methode `Utils.getRandomInteger(int min, int max)` erzeugt.

Hinweise (für die gesamte Hausübung):

- Sie können das Spiel jederzeit über die `main`-Methode der Klasse `Main` starten. Hierbei können Sie auch die Spielgeschwindigkeit anpassen. (muss für ihre Abgabe nicht zurückgesetzt werden)
- Zum legen und aufheben von Münzen sollen die Methoden `putCoin` und `pickCoin` verwendet werden.
- Ob eine Vorwärtsbewegung möglich ist soll mit der Methode `isFrontClear` überprüft werden.
- Ob eine Münze auf dem aktuellen Feld liegt soll mit der Methode `isOnACoin` überprüft werden.
- Die Roboter sollen **exakt** die vorgegebenen Schritte ausführen. Abweichungen führen zu Punktabzug.
- Ihre Implementierung muss auch dann funktionieren, wenn die Weltgröße geändert wird.

Einleitung

Das Konzept dieser Hausübung ist von einer Szene aus dem Film „Wall-E“ inspiriert, in der ein Putzroboter versucht, die dreckigen Reifenspuren, die Wall-E hinterlässt, zu reinigen und ihm hinterher zu putzen. Unser Ziel ist es, ein ähnliches Szenario unter der Verwendung des FOPBot-Frameworks zu simulieren.

Die Welt ist ein Labyrinth bestehend aus Wänden, wobei jedes Feld im Labyrinth erreichbar ist, aber die Bewegungsfreiheit durch die Platzierung der Wände eingeschränkt ist.

Es gibt zwei Arten von Robotern, die „Contaminants“ (Klassen `Contaminant1.java` und `Contaminant 2.java`) und den „Cleaner“ (Klasse `CleaningRobot.java`). Die Contaminants bewegen sich nach einem vordefinierten Muster im Labyrinth und „verschmutzen“ dabei die Welt, indem sie Münzen ablegen. Der Cleaner wird durch Tastatureingaben gesteuert. Er kann Münzen aufnehmen und ablegen, jedoch maximal 25 Münzen gleichzeitig tragen. Die Aufgabe des Cleaners besteht darin, so viele Münzen wie möglich in ein vordefiniertes, gelb markiertes Feld zu bringen, das wir als Abladezone bezeichnen. Die Abladezone befindet sich immer bei `(0, World.getHeight() - 1)`.

Das Spiel hat klare Gewinnbedingungen: Die Contaminants gewinnen, wenn mindestens 50% der Spielfläche mit Münzen belegt ist. Der Cleaner hingegen gewinnt, wenn er entweder 200 Münzen in die Abladezone gelegt hat oder wenn den Contaminants die Münzen ausgehen. (diese schalten sich dann selbst aus)

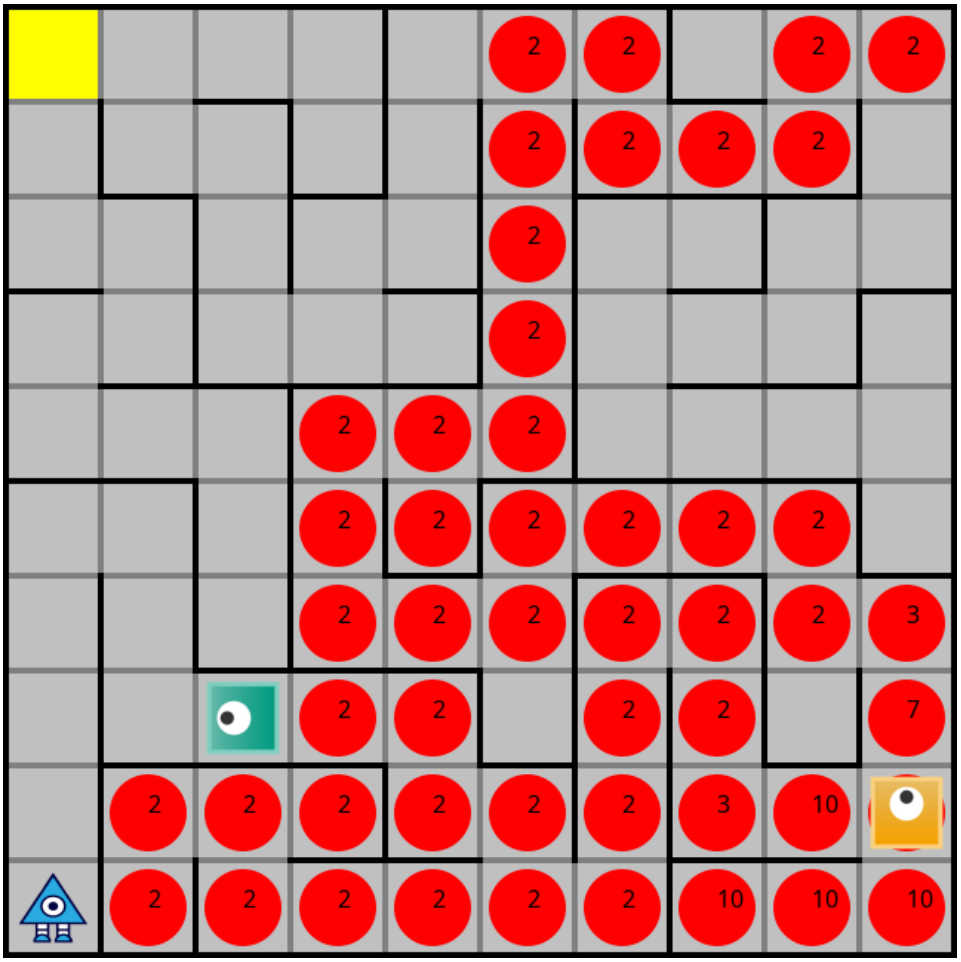


Abbildung 1: Beispiel für eine Spielsituation

H1: Steuerung des „Cleaner“-Roboters 4 Punkte

Zunächst wollen wir die Steuerung des „Cleaner“-Roboters implementieren. Der Roboter kann über Tasteneingaben gesteuert werden. Die Tastenbelegung ist wie folgt:

- `Pfeiltasten`: Roboter bewegt sich in die entsprechende Richtung
- `Keine Taste`: Roboter bleibt stehen
- `Leertaste`: Roboter legt eine Münze ab
- `R`: Roboter nimmt eine Münze auf

Die Tastatureingaben werden schon automatisch von der Vorlage abgefangen, Sie müssen diese also nicht selbst implementieren. Ihre Aufgabe ist es nun, eine Methode zu vervollständigen, die dann die Steuerung des Roboters möglich macht.

Vervollständigen Sie die Methode `handleKeyInput` der Klasse `CleaningRobot` (Datei CleaningRobot.java). Diese bekommt drei Werte übergeben:

- Parameter `k`: eine Ganzzahl, die für eine vom Spieler gedrückte Taste auf der Tastatur steht. Die Pfeiltaste nach oben hat den Wert 0, die Pfeiltaste nach rechts den Wert 1, die Pfeiltaste nach unten den Wert 2 und die Pfeiltaste nach links den Wert 3.
- Parameter `shouldPutCoins`: ein boolescher Wert, der angibt, ob der Roboter eine Münze ablegen soll.
- Parameter `shouldPickCoins`: ein boolescher Wert, der angibt, ob der Roboter eine Münze aufheben soll.

Die Methode führt drei Schritte in genau der gegebenen Reihenfolge aus:

1. Münzen ablegen, bzw. aufheben, sofern die entsprechenden Parameter gesetzt sind.
2. Drehung des Roboters in die entsprechende Richtung, sofern direction zwischen 0 und 3 liegt (beide inklusive).
3. Bewegung des Roboters in die entsprechende Richtung, sofern der Weg nicht durch eine Wand versperrt ist.

Falls `shouldPutCoins` den Wert `true` hat, soll der Roboter eine Münze auf dem aktuellen Feld ablegen, wenn er noch Münzen hat. Falls `shouldPickCoins` den Wert `true` hat, soll der Roboter eine Münze aufheben, sofern sich eine auf dem aktuellen Feld befindet. Der `Cleaner` kann maximal 25 Münzen tragen. Wenn `shouldPickCoins` den Wert `true` hat und sich eine Münze auf dem Feld befindet, der Roboter aber bereits 25 Münzen trägt, so wird die Münze einfach liegen gelassen. Sie dürfen davon ausgehen, dass `shouldPutCoins` und `shouldPickCoins` nie gleichzeitig den Wert `true` haben.

Sollte der Methode ein `direction`-Wert übergeben werden, der nicht zwischen 0 und 3 liegt (beide inklusive), so soll der `Cleaner` sich nicht bewegen. Liegt der Wert im vorher genannten Bereich, so soll der Roboter zunächst in die Richtung blicken, die der gedrückten Pfeiltaste entspricht (auch wenn er dann in Richtung einer Wand blicken würde). Also UP bei gedrückter Pfeiltaste nach oben usw. Nachdem der Roboter in die entsprechende Richtung blickt, soll er einen Schritt in eben diese Richtung gehen, sofern der Weg nicht durch eine Wand versperrt ist. Ist der Weg versperrt, so verbleibt der Roboter an seiner momentanen Position.

> **Hinweis:**
>
> in den Verbindliche Anforderungen für die gesamte Hausübung finden Sie neben den verbotenen Methoden auch einige nützliche Methoden, die Sie verwenden können.

H2: Steuerung der „Contaminant“-Roboter 8 Punkte

Nun wollen wir die Contaminants implementieren. Es gibt zwei verschiedene Arten von Contaminants. Die erste Art wählt ihre Richtung zufällig aus und ist etwas langsamer, kann dafür aber mehr Münzen ablegen. Die zweite Art hingegen wählt ihre Richtung nicht zufällig aus, sondern versucht, sich immer an der linken Wand zu orientieren. Sie ist etwas schneller, kann dafür aber immer nur zwei Münzen ablegen.

Hinweise (Für die gesamte H2**):**

- Für das generieren einer Zufallszahl wird die Methode `Utils.getRandomInteger(int min, int max)` verwendet. Diese Methode gibt eine Zufallszahl zwischen `min` und `max` (beide inklusive) zurück.
- Um Abzufragen, wie viele Münzen auf einem Feld liegen, kann die Methode `Utils.getCoinAmount(int x, int y)` verwendet werden. Diese Methode gibt die Anzahl der Münzen auf dem Feld mit den Koordinaten `x` und `y` zurück.

H2.1: Contaminant1 4 Punkte

Vervollständigen Sie die parameterlose Methode `doMove` der Klasse `Contaminant1` (Datei Contaminant1.java). Die Methode führt die folgenden Schritte in genau der gegebenen Reihenfolge aus:

1. Falls der Roboter keine Münzen mehr besitzt, so wird er ausgeschaltet.
2. Falls der Roboter ausgeschaltet ist oder keine Münzen mehr hat, so wird die Methode abgebrochen. (also die restlichen Schritte werden nicht ausgeführt)
3. Der Roboter bestimmt eine Zufallszahl mittels `Utils.getRandomInteger`, zwischen 1 und 5 (beide inklusive), und legt entsprechend viele Münzen ab. Dabei dürfen maximal 20 Münzen auf einem Feld liegen. Wenn also 20 Münzen oder mehr auf dem Feld liegen werden keine weiteren Münzen gelegt. Falls der Roboter weniger Münzen hat, als er ablegen möchte, so legt er alle Münzen ab, die er hat. (Hier wird die Methode nicht abgebrochen, falls der Roboter keine Münzen mehr hat)
4. Der Roboter dreht sich einmal gegen den Uhrzeigersinn um die eigene Achse und prüft dabei in jeder Richtung, ob er sich bewegen kann. Dannach hat er die gleiche Blickrichtung wie zuvor.
5. Der Roboter wählt aus den Richtungen, in die er sich bewegen kann, eine zufällige Richtung aus und dreht sich in diese Richtung.
6. Der Roboter bewegt sich in die entsprechende Richtung, sofern der Weg nicht durch eine Wand versperrt ist. Ist der Weg versperrt, so verbleibt der Roboter an seiner momentanen Position.

H2.2: Contaminant2 4 Punkte

Vervollständigen Sie die parameterlose Methode `doMove` der Klasse `Contaminant2` (Datei Contaminant2.java). Die Methode führt die folgenden Schritte in genau der gegebenen Reihenfolge aus:

1. Falls der Roboter keine Münzen mehr besitzt, so wird er ausgeschaltet.

2. Falls der Roboter ausgeschaltet ist oder keine Münzen mehr hat, so wird die Methode abgebrochen.

3. Wenn auf dem Feld, auf dem sich der Roboter befindet, weniger als zwei Münzen liegen, dann erhöht der Roboter die Anzahl der Münzen auf zwei. Ansonsten wird die Anzahl der Münzen auf dem Feld nicht verändert. Falls der Roboter weniger Münzen hat, als er ablegen möchte, so legt er alle Münzen ab, die er hat. (Hier wird die Methode nicht abgebrochen, falls der Roboter keine Münzen mehr hat)

4. Der Roboter dreht sich einmal gegen den Uhrzeigersinn um die eigene Achse und prüft dabei in jeder Richtung, ob er sich bewegen kann. Dannach hat er die gleiche Blickrichtung wie zuvor.

5. Jetzt orientiert sich der Roboter an der linken Wand. Dazu dreht er sich in die erste Richtung, in die er sich bewegen kann (beginnend mit der Richtung, die sich links von seiner aktuellen Blickrichtung befindet, weiter im Uhrzeigersinn). Falls er sich in keine Richtung bewegen kann, so verbleibt der Roboter an seiner momentanen Position und behält seine Blickrichtung bei.

6. Der Roboter bewegt sich in die entsprechende Richtung, sofern der Weg nicht durch eine Wand versperrt ist. Ist der Weg versperrt, so verbleibt der Roboter an seiner momentanen Position.

Hinweis:

Wenn sich der Roboter „im Urzeigersinn“ drehen soll, so muss er sich dreimal gegen den Uhrzeigersinn drehen, da es keine `turnRight` Methode für `Robots` gibt.

H3: Gewinnbedingungen **4 Punkte**

Als letztes wollen wir nun die Gewinnbedingungen überprüfen, um das Spiel zu vervoillständigen. Vervollständigen Sie dazu die Methode `checkWinCondition` der Klasse `GameController` (Datei GameController.java). Die Methode wird automatisch nach jedem Spielschritt (tick) aufgerufen und überprüft, ob eine Partei gewonnen hat. Zur Erinnerung, hier nochmal die zwei Gewinnbedingungen:

- Der `Cleaner` hat gewonnen, wenn alle `Contaminants` ausgeschaltet sind oder sich in der Abladezone mindestens 200 Münzen befinden.
- Die `Contaminants` hingegegen haben gewonnen, wenn mindestens 50% der Felder mit Münzen belegt sind, oder mathematisch ausgedrückt: $\frac{\text{Anzahl der Felder mit Münzen}}{\text{Anzahl der Felder insgesamt}} \geq 0.5$.

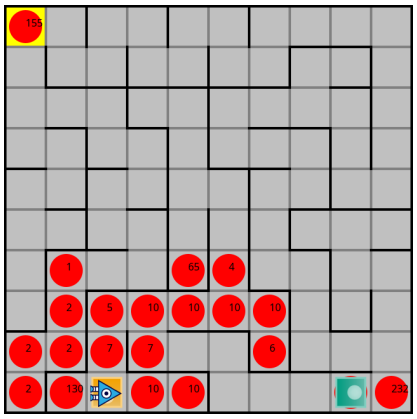
In beiden Gewinnfällen sollen die Roboter der unterlegenen Partei ausgeschaltet werden, die Gewinnnachricht in die Konsole ausgegeben werden und die Methode `stopGame()` aufgerufen werden. Die Gewinnnachricht soll dabei folgendermaßen aussehen:

- `Cleaner` gewinnt: `"Cleaning robot won!"`
- `Contaminants` gewinnen: `"Contaminants won!"`

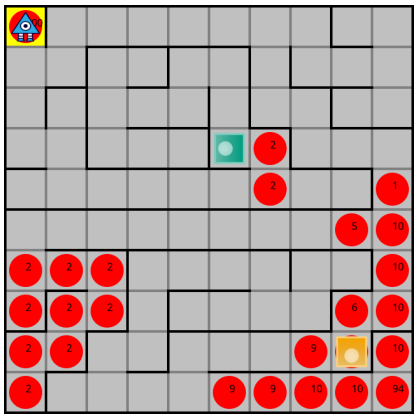
Falls beide Parteien gleichzeitig gewinnen, so soll der `Cleaner` gewinnen.

Hinweise:

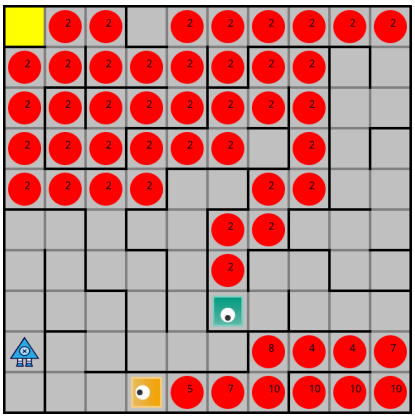
- Die Instanzen der drei Roboter erhalten Sie über die Methoden `getCleaningRobot()`, `getContaminant1()` und `getContaminant2()`.
- Nutzen Sie wieder die Methode `Utils.getCoinAmount(int x, int y)`, um die Anzahl der Münzen auf einem Feld zu erhalten.



(a) Cleaner gewinnt, da alle Contaminants ausgeschaltet sind und ihre Münzen verbraucht haben.



(b) Cleaner gewinnt, da er mehr als 200 Münzen in die Abladezone gelegt hat.



(c) Contaminants gewinnen, da mehr als 50% der Felder mit Münzen belegt sind.

Abbildung 2: Beispiele für Gewinnbedingungen