Funktionale und objektorientierte Programmierkonzepte

Kim Berninger, Jan Bormet, Dustin Glaser,
Julius Hardt und Alexander Mainka
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 20/21
v1.3.2

Übungsblatt 2

Themen: FopBot, Referenzsemantik, Klassen, Arrays

Relevante Foliensätze: 01a bis 01e

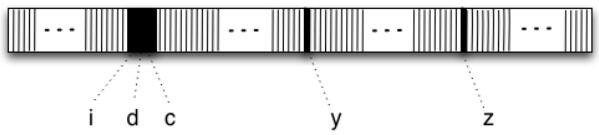
Abgabe der Hausübung: 27.11.2020 bis 23:50 Uhr

V Vorbereitende Übungen

V1 Referenzen ☆ ☆ ☆

Geben Sie in eigenen Worten wieder, was man unter einer Referenz versteht.

Betrachten Sie außerdem folgendes Schaubild und den Codeausschnitt aus der Vorlesung:




```java
public class X{
    int i;
    double d;
    char c;
    ...
}
```

Zeichnen Sie die Referenzpfeile nach den folgenden Aufrufen ein (ergänzen Sie auch die neuen Reservierungen des Speicherplatzes, wenn nötig):

```java
X y = new X();
X z = y;
y = new X();
```

V2 Zuweisen und Kopieren ☆ ☆ ☆

Erläutern Sie in Ihren eigenen Worten den Unterschied zwischen Zuweisen und Kopieren. In welchen Fällen sind beide Aktionen synonym zu betrachten?

Wie können Sie eine Zuweisung beziehungsweise eine Kopie in Java umsetzen? Nennen Sie jeweils ein Beispiel.

V3 Arrays ★ ☆ ☆

Welche Aussagen zu einem gegebenen Array `a` sind wahr?

(1) Alle Einträge des Arrays müssen vom selben Typ sein.

(2) Ein Array hat keine feste Größe und es können beliebig viele neue Einträge einem Array hinzugefügt werden.

(3) Um die Anfangsadresse einer Komponente an Index i zu bekommen, wird i-mal die Größe einer Komponente auf die Anfangsadresse von `a` addiert.

(4) Außer den eigentlichen Komponenten des Arrays enthält das Arrayobjekt nichts weiteres.

(5) Ein Array kann nur primitive Datentypen wie zum Beispiel `int`, `char` oder `double` speichern. Somit ist es insbesondere nicht möglich, `Robot`-Objekte in einem Array zu speichern.

Schreiben Sie die nötigen Codezeilen (auf Papier), um ein Array `a` der Größe 42 vom Typ `int` anzulegen. Füllen Sie danach das Array mithilfe einer Schleife, sodass an der Stelle `a[i]` der Wert `2i+1` steht. Nutzen Sie dabei zuerst eine while- und danach eine for-Schleife.

V4 Wettrennen ★ ☆ ☆

Sie haben einen schnellen Roboter `rabbit` erstellt und wollen ihm nun noch ein langsames Gegenstück `turtle` bauen. Beide starten an einem gemeinsamen Punkt, schauen in die gleiche Richtung und besitzen die gleiche Anzahl an Coins. Sie wollen nun schauen, wer in 10 Runden mehr Strecke zurücklegen kann. Jeder der beiden Kontrahenten kommt pro Runde genau einen Schritt voran, der schnelle `rabbit` erhält jedoch in jeder zweiten Runde einen Extraschritt. Betrachten Sie den folgenden Codeausschnitt, der die Situation implementieren möchte:

```
1  Robot rabbit = new Robot(0, 0, RIGHT, 0);
2  Robot turtle = rabbit;
3
4  for(int i = 0; i < 10; i++){
5      if(i / 2 == 0){
6          rabbit.move();
7      }
8
9      rabbit.move();
10     turtle.move();
11 }
```

Führen Sie den Code einmal selbst in Eclipse aus und schauen Sie was passiert! Beheben Sie danach **alle** vorhandenen Fehler in der Implementierung, um die oben beschriebene Situation exakt umzusetzen.

V5 Roboter miteinander vergleichen ★ ☆ ☆

Schreiben Sie eine Methode `int robotsEqual(Robot a, Robot b)`. Diese bekommt zwei Roboter übergeben und soll 2 zurückgeben, wenn die Attribute `x`, `y`, `direction` und `numberOfCoins` bei beiden Robotern die gleichen Werte haben. 1 soll zurückgegeben werden wenn sich beide Roboter nur auf demselben Feld befinden, andernfalls gibt die Methode 0 zurück.

V6 Primitive Datentypen ★ ★ ☆

Schreiben Sie eine Methode `char smallestPDT(long n)`. Diese bekommt eine ganze Zahl übergeben und soll den primitiven Datentyp zurückgeben, der den wenigsten Speicherplatz verbraucht, aber immer noch die übergebene Zahl speichern kann. Geben sie `'l'` für den Datentyp `long` zurück, `'i'` für `integer` usw.

Schreiben Sie nun eine Methode `char[] smallestPDTs(long[] a)`. Diese bekommt ein Array von ganzen Zahlen übergeben und soll ein Array zurückgeben, dass die Methode `smallestPDT(long n)`, in der gleichen Reihenfolge wie in `a`, auf jede Zahl in `a` anwendet.

V7 Aufsummieren ★ ★ ☆

Schreiben Sie eine Methode `void sumUp(int[] a)`, die ein Array `a` von Typ `int` erhält.

An Index $i \in$ `{0,...,a.length-1}` in `a` soll nun der neue Wert `a[0]+...+a[i]` geschrieben werden. Dabei bezeichnen `a[0],...,a[i]` die Werte in `a` unmittelbar vor dem Aufruf der Methode.

Übergeben wir der Funktion das folgende Array `a = [3, 4, 1, 9, -5, 4]`, so wird das Array folgendermaßen modifiziert:

→ `[3, 3+4, 3+4+1, 3+4+1+9, 3+4+1+9+(-5), 3+4+1+9+(-5)+4]`

→ `[3, 7, 8, 17, 12, 16]`

V8 Liste von Positionen ★ ★ ★

In dieser Aufgabe werden wir ein wenig kreativ und zeichnen die Initialen der `FOP` mit dem FOP-Bot. Dazu müssen wir zunächst sicherstellen, dass unser Roboter eine beliebige gegebene Position automatisiert ansteuern kann.

Um uns die Arbeit zu vereinfachen, verwenden wir die Klasse `Point` der Java-Standardbibliothek, deren Instanzen Punkte im zweidimensionalen Raum repräsentieren. Ein `Point`-Objekt kann mittels des Konstruktors `Point(int x, int y)` erzeugt werden. Die Abfrage der Werte ist dann über die Objekt-Attribute `x` und `y` möglich.

V8.1 Setzen der Blickrichtung

Als Erstes soll die `public`-Methode `void setDirection(Robot robot, Direction direction)` implementiert werden: Diese bekommt ein `Robot`-Objekt sowie eine `Direction`-Konstante übergeben. Nach Aufruf der Methode soll der Roboter in die gewünschte Richtung blicken.

V8.2 Bewegen zu einer Position

Implementieren Sie nun die `public`-Methode `void moveToPoint(Robot robot, Point point)`: Mit dem Aufruf der Methode soll der gegebenen Roboter mittels der soeben von Ihnen implementierten Methoden `setDirection` und der Ihnen bereits bekannten Methode `move` an die gegebene Position bewegen werden. Sie können davon ausgehen, dass sich auf dem Weg zu dieser Position keine Hindernisse befinden. Dabei ist nicht wichtig, dass der Roboter den kürzesten Weg findet.

V8.3 Coin Patterns

Nun implementieren Sie die `public`-Methode `void putCoins(Robot robot, Point[] points)`: Der gegebene Roboter soll an jeder der im gegeben Array enthaltenen Position eine Münze ablegen, sich anschließend in die Mitte der Welt bewegen und nach oben blicken.

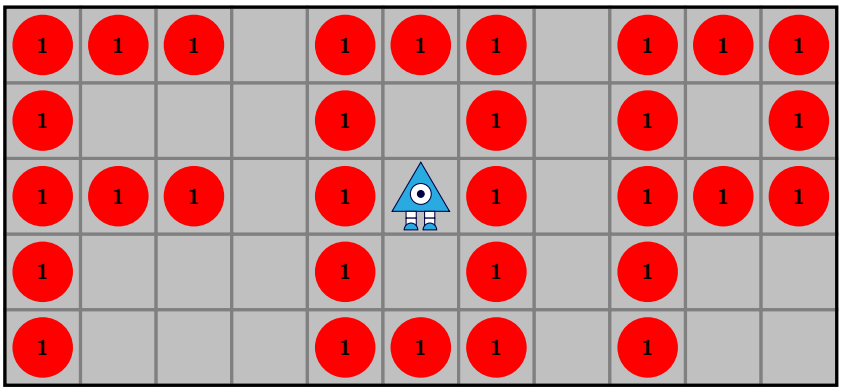


Abbildung 1: Gewünschtes Ergebnis

Verbindliche Anforderung: Die einzelnen Positionen des `Point`-Arrays müssen in einer einzigen `for`-Schleife abgelaufen werden. Die Objektmethode `putCoin` der Klasse `Robot` darf nur innerhalb dieser `for`-Schleife aufgerufen werden.

Autor V8: Ruben Deisenroth

H Zweite Hausübung — Gesamt 24 Punkte
Tetris

In dieser Hausübung realisieren Sie eine leicht abgeänderte Version des Spiele-Klassikers Tetris unter Verwendung des FopBot-Frameworks.

Ziel von Tetris ist es, einen möglichst hohen Punktestand zu erreichen, indem nacheinander herabfallende Steine so angeordnet werden, dass diese ausgefüllte Reihen bilden, welche sodann verschwinden. Spieler/innen haben die Möglichkeit, die Steine während des Falls nach links und rechts zu bewegen und schrittweise zu rotieren. Berührt ein Stein mit seiner Unterseite den Boden oder einen anderen Stein, fixiert dieser sich nach kurzer Zeit und ein neuer Stein erscheint. Sobald die Fläche so weit ausgefüllt ist, dass keine weiteren Steine mehr platziert werden können, endet das Spiel.

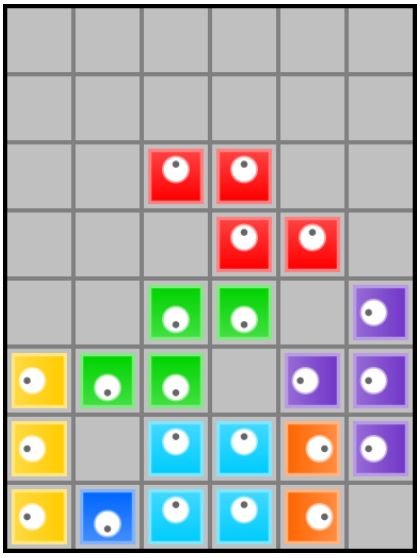


Abbildung 2: FopBot-Tetris mit Square-Robotern

Aufbau der Vorlage

In der Vorlage sind einige Bestandteile des Spiels bereits implementiert. Damit das Spiel aber benutzbar wird, muss diese Vorlage von Ihnen ergänzt werden.

Die sich wiederholende Abfolge zwischen Stein-Generierung und Entfernen aller ausgefüllten Reihen wird in dieser Hausübung Iteration genannt. Die maximale Iterationsanzahl ist gleich der maximalen Steinanzahl, welche in der Klasse `Main` in der Konstanten `STONE_COUNT` festgelegt ist. In derselben Klasse finden Sie auch die Konstanten `WIDTH` und `HEIGHT`, welche die Breite beziehungsweise Höhe des Spielfelds in Feldern angeben. Die Werte dieser Konstanten dürfen von Ihnen auf Wunsch geändert werden. Legen Sie einen beliebigen Integer-Wert für die Konstante `SEED` fest, wenn Sie in jeder Runde die gleiche Abfolge von Steinen erhalten möchten. Dies könnte beispielsweise zum Beheben möglicher Fehler nützlich sein. Mit dem Standardwert `null` variiert die Steinabfolge von Spiel zu Spiel.

Die abstrakte Klasse `Stone` modelliert den wohl wichtigsten Teil des Puzzle-Klassikers: die Spielsteine. Objekte konkreter `Stone`-Klassen verwalten eine beliebig große, aber nicht leere Menge an Square-Robotern, welche als bunte, einäugige Quadrate auf dem Spielfeld erscheinen und im Team als Stein fungieren.

Die Original-Spielsteine I, J, L, O, S, Z und T, die man wegen ihres Aufbaus aus vier Quadraten auch Tetrominos nennt, werden mittels Objekten der gleichnamigen Klassen dargestellt, die jeweils vier Square-Roboter verwalten, welche zusammen den entsprechenden Spielstein darstellen.

Abbildung 3: Z-Tetromino aus vier Square-Robotern mit Koordinaten

In der Klasse `Stone` sind unter anderem die Methoden `moveAllDown()`, `moveAllLeft()` und `moveAllRight()` gegeben. Beim Aufruf dieser wird zunächst überprüft, ob alle zugehörigen Square-Roboter um einen Schritt nach unten, nach links beziehungsweise nach rechts bewegt werden können. Ist dies der Fall, werden alle zugehörigen Square-Roboter um einen Schritt nach unten, nach links beziehungsweise nach rechts bewegt und die Methode gibt `true` zurück. Im anderen Fall werden keine Bewegungen ausgeführt und die Methode gibt `false` zurück. Zudem ist die Methode `getSquareRobots()` gegeben: Diese gibt ein Array des Komponententyps `SquareRobot` zurück, welches alle dem Stein zugehörigen Square-Roboter enthält. Die Methode `getRelatedGame()` liefert das `TetrisGame`-Objekt, das den Stein verwendet.

In der Klasse `SquareRobot` sind unter anderem die Methoden `moveDown()`, `moveLeft()` sowie `moveRight()` gegeben. Beim Aufruf dieser bewegt sich der einzelne Square-Roboter einen Schritt nach unten, nach links beziehungsweise nach rechts ohne sicherzustellen, dass die Bewegung möglich ist. Wegen ausgefüllter Reihen zu entfernende Square-Roboter werden mittels der Methode `turnOff()` ausgeschaltet. Solche sind dann auf dem Spielfeld nicht mehr sichtbar. Ob ein Roboter ausgeschaltet wurde, kann unter anderem mit der Methode `isTurnedOff()` abgefragt werden. In welcher Iteration ein Roboter ausgeschaltet wurde, lässt sich mit der Methode `getTurnOffIteration()` abfragen: Diese gibt die Iteration zurück oder `-1`, wenn dieser noch nicht ausgeschaltet wurde. Die Methode `getRelatedStone()` liefert das zugehörige `Stone`-Objekt des Square-Roboters. Die Klasse `SquareRobot` besitzt darüber hinaus auch alle Methoden der Ihnen bereits bekannten Klasse `Robot`.

Die Klasse `TetrisGame` bietet mehrere Methoden, die für die Implementierung des Spiels notwendig sein könnten: Die Methode `getStoneArray()` gibt ein `Stone`-Array zurück, welches alle bisher erzeugten `Stone`-Objekte enthält und darüber hinaus Platz für weitere bietet. Dieses Array darf nicht modifiziert werden. Die Methoden `getWidth()` und `getHeight()` geben die Breite beziehungsweise Höhe des Spielfelds zurück. Das Spiel lässt sich über die Klasse `Main` starten.

Verbindliche Anforderung: Für diese Hausübung dürfen keine anderen Datenstrukturen als Arrays verwenden. Des Weiteren verboten sind Hilfsmethoden aus der Java-Standardbibliothek.

H1 Spielfeld 7 Punkte

In dieser Aufgabe implementieren Sie zwei Hilfsmethoden, von denen in der Vorlage bereits eine Verwendung findet. Darüber hinaus dürfen Sie natürlich auch eigene Hilfsmethoden implementieren. Nutzen Sie diese Möglichkeit gerne, um Code-Redundanz zu vermeiden und Ihren Arbeitsaufwand zu reduzieren!

H1.1 Mögliche Verschiebungen 4 Punkte

Zunächst soll die Möglichkeit gegeben werden, für die aktuelle Position eines Square-Roboters **alle** möglichen Verschiebungen auf dem Spielfeld abzufragen. Eine Verschiebung um den Vektor $\vec{v} = (v_1, v_2)^T$ ist für die aktuelle Position $\vec{p} = (p_1, p_2)^T$ genau dann möglich, wenn die dadurch resultierende Position $\vec{p}_{new} = (p_1 + v_1, p_2 + v_2)^T$ eine nicht-belegte Position auf dem Spielfeld ist. Dabei ist nicht relevant, ob der Weg zu der neuen Position „versperrt“ ist. Eine Position auf dem Spielfeld wird als belegt bezeichnet, wenn sich auf dieser ein eingeschalteter Square-Roboter eines anderen Steins befindet. Eine Verschiebung wird mittels eines Objekts der in der Vorlage gegebenen Klasse `Vector` dargestellt.

Implementieren Sie dazu die Methode `getPossibleMovements()` in der Klasse `SquareRobot`: Diese gibt ein vollständig ausgefülltes Array des Komponententyps `Vector` zurück, das alle möglichen Verschiebungen, dargestellt als `Vector`-Objekte, jeweils ein Mal enthält. Die Verschiebung um den Vektor $(0, 0)^T$ soll nicht im Array enthalten sein.

Die Klasse `Vector` kann genauso wie die Ihnen aus Aufgabe V8 bekannte Klasse `Point` verwendet werden: Ein `Vector`-Objekt wird mittels des Konstruktors `Vector(int x, int y)` erzeugt. Die Abfrage der Werte ist dann auch über die Objekt-Attribute `x` und `y` möglich. Darüber hinaus dürfen Sie auch alle anderen Methoden der Klasse `Vector`[1] verwenden.

H1.2 Verschiebbarkeit 3 Punkte

Des Weiteren soll für einen Square-Roboter eine einfache Möglichkeit gegeben werden abfragen zu können, ob dieser um einen gegebenen Vektor verschoben werden kann.

Implementieren Sie dazu die Methode `canMove(Vector vector)` in der Klasse `SquareRobot`: Diese bekommt ein `Vector`-Objekt übergeben und gibt genau dann `true` zurück, wenn eine Verschiebung um den Vektor möglich ist. Andernfalls wird `false` zurückgegeben. Für den Vektor $(0, 0)^T$ soll sinnvollerweise in jedem Fall `true` zurückgegeben werden.

H2 Rotieren 6 Punkte

Die Rotationen der implementierten Tetromino-Steine erfolgen weitestgehend nach den Original-Regeln[2], die bereits in der Vorlage definiert sind und in den entsprechenden Methoden nur noch verwendet werden müssen.

[1] Dokumentation Klasse `Point`: https://docs.oracle.com/.../Point.html
[2] Originales Rotationssystem: https://tetris.wiki/Original_Rotation_System

Für diese Implementierung wird die Funktionalität der Roboter-Rotation so erweitert, dass neben der Änderung der Blickrichtung zusätzlich eine neue Position auf dem Spielfeld eingenommen wird, die so gewählt ist, dass mit dem Rotieren aller Square-Roboter eines Steins eine Linksdrehung dieses Steins umgesetzt wird.

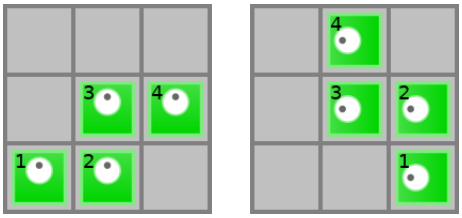


Abbildung 4: S-Tetromino vor und nach der ersten Linksrotation

H2.1 Rotieren einzelner Square-Roboter 3 Punkte

Die Methode `getRotationVectors()` gibt ein `Vector`-Array zurück, das für eine Anzahl n an bereits ausgeführten Stein-Rotationen an Stelle n eine Bewegungsvorschrift enthält, die von diesem Square-Roboter für die nächste Stein-Rotation ausgeführt werden muss. Nachdem die letzte Bewegungsvorschrift eines Square-Roboters erreicht wurde, beginnt die Zählung für diesen Square-Roboter von vorne. Die Array-Größe ist gleich der Anzahl der Bewegungsvorschriften, die ausgeführt werden können, bis die Zählung von vorne beginnt. Bei jedem Aufruf soll zudem eine normale Roboter-Drehung stattfinden. Die im Array enthaltenen `Vector`-Objekte sind wie in Aufgabe H1.1 zu verstehen.

Implementieren Sie dafür die Methode `rotateLeft()` in der Klasse `SquareRobot`, die eine Rotation für diesen umsetzt, sofern eine Bewegungsvorschrift gegeben ist. Verwenden Sie zur Positionierung die Methoden `setX(int x)` und `setY(int y)` oder die Methode `setField(int x, int y)`. An dieser Stelle darf ohne Überprüfung davon ausgegangen werden, dass der Square-Roboter um den entsprechenden Vektor verschoben werden kann.

Beispielsweise wurden für den in Abbildung 4 gezeigten Stein folgende Bewegungsvorschriften verwendet:

Roboter	Bewegungsvorschrift	Position vorher	Position nachher
1	`(2,0)`	`(0,0)`	`(2,0)`
2	`(1,1)`	`(1,0)`	`(2,1)`
3	`(0,0)`	`(1,1)`	`(1,1)`
4	`(-1,1)`	`(2,1)`	`(1,2)`

H2.2 Rotieren ganzer Steine 3 Punkte

Implementieren Sie nun die Methode `rotateAllLeft()` in der Klasse `Stone`.

Diese Methode ruft die Methode `rotateLeft()` für alle zugehörigen Square-Roboter des Steins auf, sofern für alle Square-Roboter die jeweilige Bewegungsvorschrift ausgeführt werden kann. Ist dies der Fall, gibt die Methode `true` zurück. Andernfalls wird `false` zurückgegeben.

H3 Steuerung 2 Punkte

Damit ein Stein gesteuert werden kann, muss dieser auf Eingaben reagieren können.

Implementieren Sie nun die Methode `handleKeyInput(byte key)` in der Klasse `Stone`. Diese bekommt einen Wert `key` übergeben und führt die dafür vorgesehene Handlung aus:

- `0`: eine Rotation nach links
- `1`: eine Bewegung nach links
- `2`: eine Bewegung nach unten
- `3`: eine Bewegung nach rechts
- `4`: so viele Bewegungen nach unten wie möglich

Für weitere Eingabewerte soll keine Reaktion erfolgen.

Nach Implementierung dieser Methode ist eine intuitive Steuerung über die Pfeiltasten sowie über die Tasten `W`, `A`, `S` und `D` möglich, wobei ↑ (beziehungsweise `W`) eine Rotation nach links auslöst. Das Betätigen der Leertaste löst ein Herabfallen des Steins aus.

Beachten Sie die in der Einleitung genannten zur Verfügung stehenden Methoden.

H4 Entfernen und Nachrücken 5 Punkte

Nach dem Legen eines Steins soll überprüft werden, ob sich durch diesen ausgefüllte Reihen ergeben haben. Ist dies der Fall, werden die Square-Roboter dieser Reihen ausgeschaltet. Für eine ausgeschaltete Reihe werden alle nachfolgenden Reihen nachgerückt, indem alle eingeschalteten Square-Roboter nachfolgender Reihen um einen Schritt nach unten verschoben werden. Besitzt eine Reihe mehrere vorangehende ausgeschaltete Reihen, wird sie um die Anzahl dieser Reihen verschoben.

Implementieren Sie dazu die Methode `clearRows()` in der Klasse `TetrisGame`, welche dieses Verhalten umsetzt. Nach dem Entfernen und Nachrücken gibt die Methode die Anzahl der entfernten Reihen zurück.

H5 Scoring 4 Punkte

Zum Abschluss implementieren Sie ein Punktesystem, um Spielern ein Feedback zum Verlauf des Spiels zu geben zu geben:

Spieler bekommen für x in einer Iteration ausgefüllte Reihen immer zweimal so viele Punkte wie für $x-1$ Reihen, wobei eine Reihe 1.000 Punkte einbringt. Jede Reihe, in der alle Square-Roboter gleich ausgerichtet sind, wird zweimal gezählt, wobei wir mit x_{total} die Anzahl der tatsächlich gewerteten Reihen bezeichnen.

Zur Berechnung der Punkte kann die folgende mathematische Funktion verwendet werden:

$$f(x_{total}) = \begin{cases} 0 & \text{wenn } x = 0 \\ 1000 & \text{wenn } x = 1 \\ 2 \cdot f(x-1) & \text{sonst} \end{cases}$$

Beispielsweise bringen vier in einer Iteration ausgefüllte Reihen, in denen die Roboter der letzten beiden Reihen gleich ausgerichtet sind, $f(1 + 1 + 2 + 2) = f(6) = 32.000$ Punkte.

Implementieren Sie dazu die Methode `getPoints(int iteration)` in der Klasse `TetrisGame`: Diese bekommt eine Iteration übergeben, berechnet die Anzahl der in dieser Iteration erreichten Punkte und gibt das Ergebnis zurück. Beachten Sie bei der Bearbeitung dieser Aufgabe unbedingt die in der Einleitung genannten Methoden!