# Allgemeine Informatik 2
# Wintersemester 2019/2020

TECHNISCHE UNIVERSITÄT DARMSTADT

**Programmierprojekt, Version: 10. Dezember 2019**
Bearbeitungszeit: 01.12.2019 bis 17.01.2020

## Organisatorisches

**Wichtig:** Lesen Sie sich zunächst die folgenden Formalitäten genau durch. Sollten diese nicht exakt beachtet werden, können wir Ihre Abgabe unter Umständen nicht werten. Lesen Sie die Aufgabenstellung vollständig durch, bevor Sie mit der Implementierung beginnen.

- Beachten Sie alle Hinweise zum Programmierprojekt auf der Seite:
  `https://moodle.informatik.tu-darmstadt.de/course/view.php?id=759`
- Nutzen Sie unsere **Vorgabe** und tragen Sie in der enthaltenen Datei **README.txt** Ihren Namen, Ihre Matrikelnummer und Ihre TU-ID (sowie bei Zweiergruppen die Daten Ihrer Partner*in!) ein.
- Bei Zweiergruppen reicht es aus, wenn die Lösung nur *einmal* hochgeladen wird. Sie sollten aber die Co-Autor*in in Moodle kennzeichnen.
- Abgabe des Projekts (spätestens bis zum **17.01.2020** um **12:00 Uhr**)
  - Verpacken Sie Ihr komplettes Projektverzeichnis in eine Standard-ZIP-Datei. Der Name muss aus den Initialen der Gruppenmitglieder und der Umgebung mit welcher Sie gearbeitet haben bestehen, also z. B. **MK_bluej.zip** bzw. **MK_eclipse.zip**. Bei Zweiergruppen entsprechend beide Namen, also z. B. **`MK_CK_bluej.zip`**.
  - **Keine** ZipX-, 7z- oder RAR-Archive!
  - Laden Sie die ZIP-Datei auf unserer moodle-Plattform hoch.
  - Probieren Sie die Abgabe auf jeden Fall schon deutlich vor dem Abgabeschluss aus! Sie können mehrfach abgeben – bei mehrfachen Abgaben wird nur die zeitlich Letzte gewertet.
- Es sind insgesamt 20 Punkte zu erreichen. Für Mängel bei Formatierung oder Kommentierung gibt es Abzüge (siehe Aufgabe 6). Nicht kompilierbare Abgaben werden mit **0 Punkten** bewertet. Wir gehen davon aus, dass Sie das Projekt mit **BlueJ** oder **Eclipse** bearbeiten. Sie können eine andere IDE Ihrer Wahl verwenden – das Projekt **muss** jedoch direkt unter **BlueJ** oder **Eclipse** lauffähig sein und auch die entsprechende README.txt-Datei enthalten.
- **Achtung:** Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Lösung bestätigen Sie, dass Sie die alleinige Autor*in des gesamten Materials sind. Das bedeutet nicht nur, dass „Abschreiben“ (auch verfremdet) verboten ist, sondern auch „Abschreiben lassen“ und „Lösungen vergleichen“ – egal, ob das durch Weitergabe von Programmcode oder mündlich erfolgt. Lassen Sie Ihr Notebook / Ihren Rechner / Ihren USB-Stick nicht unbeaufsichtigt. Bei Unklarheiten zu diesem Thema finden Sie weiterführende Informationen unter:

  **`https://www.informatik.tu-darmstadt.de/studium_fb20/im_studium/studienbuero/plagiarismus/index.de.jsp`**

  - Konsequenzen: **0 Punkte** im Programmierprojekt oder gar in der gesamten Prüfung!
  - Selbstverständlich nutzen wir zusätzlich zu einer manuellen Prüfung eine recht zuverlässige Plagiatschecker-Software.

## Wichtige Hinweise

1) Das Projekt ist zu Beginn noch nicht vollständig kompilierbar, dies ist so beabsichtigt! Es fehlen noch die Klassen und Methoden, welche Sie in **Aufgabe 2**, **Aufgabe 3** und **Aufgabe 4**/**Aufgabe 5** selbst schreiben sollen.

2) Das gesamte Projekt benötigt mindestens **Java Version 11**. Sie können Ihre installierte Java Version wie folgt überprüfen:

   - **Windows**[1]: *Start* → *Alle Programme* → *Zubehör* → *Eingabeaufforderung* und geben Sie Folgendes ein:

     *java -version*

   - **Linux & Mac:** Öffnen Sie ein Terminal und geben Sie *java -version* ein und drücken Sie Enter.

   Falls Sie eine veraltete Java Version besitzen, müssen Sie zuerst auf Java 11 (oder neuer) updaten. Hierfür finden Sie Download-Dateien und eine Installationsanleitung z. B. unter `https://www.oracle.com/technetwork/java/javase/downloads/jdk12-downloads-5295953.html`

   ***Hinweis:*** Wir haben die Erfahrung gemacht, dass unter **Linux** das OpenJDK 11 Probleme mit der Ausführung dieses Projekts haben kann. Sollten Sie das Spiel also nicht starten können, so versichern Sie sich, dass Sie die Java Version von Oracle installiert haben.

3) Sie können das Projekt unter der gewohnten Entwicklungsumgebung **BlueJ** bearbeiten. Gerne können Sie auch die Gelegenheit nutzen, um **Eclipse** (eine leistungsfähigere IDE) kennenzulernen und das Projekt hierin bearbeiten. Dies bietet Ihnen den Vorteil einer besseren Übersichtlichkeit. Installationsdateien und Hinweise finden Sie hierzu unter `http://www.eclipse.org/downloads`.

4) Um das Projekt in die Entwicklungsumgebung korrekt einzubinden, gehen Sie bitte wie folgt vor:

   - **Eclipse:** Laden Sie sich zunächst von unserer moodle-Plattform die Datei **propro_eclipse.zip** herunter. Öffnen Sie **Eclipse** und klicken Sie auf:

     *File* → *Import* → *General* → *Existing Projects into Workspace* → *Select archive file* → *Browse*

     Wählen Sie die eben heruntergeladene Datei **propro_eclipse.zip** aus und klicken Sie anschließend auf *Finish*.

   - **BlueJ:** Laden Sie sich zunächst von unserer moodle-Plattform die Datei **propro_bluej.zip** herunter. Entpacken Sie das Archiv und starten Sie **BlueJ**. Klicken Sie nun auf:

     *Project* → *Open Project...*

     Wählen Sie den eben entpackten Ordner aus und bestätigen Sie die Eingabe.

5) **Hinweis für Linux-Nutzer:** Sollten Sie Linux verwenden und das Projekt in BlueJ bearbeiten wollen, dann müssen Sie BlueJ mit dem manuellen Installer installieren, damit es das OracleJDK verwendet. Hierfür haben wir ein extra PDF mit den einzelnen Schritten auf Moodle zur Verfügung gestellt.

## Vorgaben

Sie sollten nun in der Lage sein, das Programmierprojekt zu öffnen. Es enthält alle benötigten Klassen in Unterpaketen des Pakets `tud.ai2.tetris`. Während des Programmierprojekts werden Sie nur in den Unterpaketen **model** und **view** arbeiten (die anderen Pakete können Sie ignorieren). Die meisten der Klassen und Methoden sind schon vollständig. Soweit es nicht explizit anders angegeben ist, sollen alle von Ihnen zu implementierenden Methoden und Klassen `public` und alle Klassen- und Instanzvariablen `private` sein. Einige Methoden sind noch nicht implementiert bzw. enthalten nur eine "Dummy-Implementierung" mit falschem Rückgabewert. Es ist im Folgenden Ihre Aufgabe diese Methoden zu vervollständigen (und auf Kommentierung sowie Formatierung zu achten).
Beachten Sie zudem, dass alle übergebenen Parameter einer Methode geprüft werden müssen, dass diese verwendet werden können, auch wenn dies nicht explizit in der Aufgabenstellung steht! Damit ist insbesondere gemeint, dass Sie eventuellen `NullPointerException`s vorbeugen sollen! In einem solchen Randfall soll eine `IllegalArgumentException` geworfen werden.

**Alle anderen Methoden und Klassen der Vorgabe müssen unangetastet bleiben, auch sind keine zusätzlichen Datenfelder jedweder Sichtbarkeit erlaubt! Zusätzliche private Hilfsmethoden sind jedoch in jedem Fall erlaubt bzw. erwünscht.**

[1] Falls Sie Windows 10 nutzen, geben Sie in die Suchleiste einfach *Eingabeaufforderung* ein.

## Einleitung - Das Spiel *Tetris*

In diesem Programmierprojekt werden Sie eine einfache Variante des bekannten Spiels „Tetris“ programmieren. In der Vorgabe (die Sie sich auf jeden Fall gut ansehen sollten) ist schon ein Grundgerüst enthalten, wichtige Teile fehlen aber noch und sind von Ihnen zu implementieren.

Folgende Beschreibung des Spielprinzips basiert auf dem Wikipedia-Artikel “Tetris”[2]:
Tetris ist ein Computerspiel des russischen Programmierers Alexei Paschitnow, der die erste spielbare Version im Juni 1984 auf einem Elektronika-60-Rechner fertigstellte. Einzeln vom oberen Rand des rechteckigen Spielfelds herunterfallende, stets aus vier quadratischen **Blöcken** zusammengesetzte **Steine** („Tetrominos“) müssen vom Spieler in 90-Grad-Schritten gedreht und so platziert werden, dass sie am unteren Rand (der **Matrix**) horizontale, möglichst lückenlose Reihen bilden. Sobald eine Reihe von Quadraten komplett ist, verschwindet sie; alle darüberliegenden Reihen rücken nach unten und geben damit einen Teil des Spielfelds wieder frei. Der Name des Spiels rührt von dem griechischen Wort für vier, tetra, und bezeichnet das gleichzeitige Tilgen von vier Reihen sowie die Zahl der Quadrate pro Form.

Das Spiel endet, sobald sich die nicht abgebauten Reihen bis zum oberen Spielfeldrand aufgetürmt haben. Das Spielprinzip lehnt sich an das Spiel Pentomino an; im Unterschied zu diesem besitzt Tetris jedoch nur fünf statt zwölf Formen. Diese Formen werden häufig mit den lateinischen Buchstaben bezeichnet, denen sie ähneln. Während **I**, **O** und **T** symmetrisch sind, gibt es bei den Formen **Z** und **L** zwei spiegelbildliche Varianten (**J** / **L** und **S** / **Z**), woraus sich die Gesamtzahl von sieben ergibt.

## Terminologie

Dieser Abschnitt erklärt wichtige Begriffe unserer Version des Spiels näher. *Abbildung 1* verdeutlicht einige Teilaspekte.

**Matrix** Die Matrix ist der Bereich des Spielfeldes, der die abgelegten **Blöcke** enthält. Sie besteht aus mehreren Zeilen und Spalten, wobei Zeilen von unten nach oben nummeriert sind,d.h.die unterste Zeile hat den Index 0.

**Block** Blöcke sind die atomaren Bestandteile der **Matrix**. Sie entstehen, wenn ein **Stein** auf die Matrix abgelegt wird. Dieser Vorgang wird im Folgenden als **manifestieren** bezeichnet.

**(Spiel-)Stein** Ein Stein ist ein Tetromino, welches noch nicht auf der Matrix **manifestiert** wurde. Es kann vom Spieler im Spielfeld nach links oder rechts bewegt oder gedreht werden. Durch die im Spiel festgelegte Gravitation fällt er automatisch.

**Manifestieren** Hierbei wird aus dem fallenden **Stein** auf der **Matrix** eine Anordnung von mehreren **Blöcken** erzeugt.

**Form** Eine Form definiert das Aussehen eines **Steins**, also dessen Farbe und welche **Blöcke** vom Stein beim Manifestieren belegt werden und welche nicht.

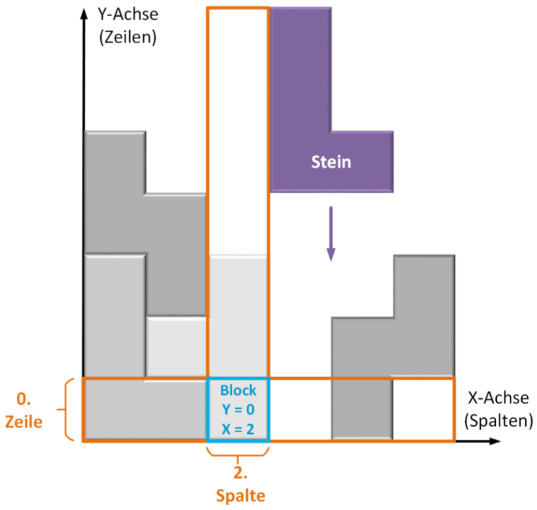



**Abbildung 1:** Terminologie

[2] `https://de.wikipedia.org/wiki/Tetris`; Zuletzt aufgerufen: 28. September 2019, 16:32 Uhr

## Aufgabe 1 GUI vervollständigen (2 Punkte)

In der Klasse `MainFrame` ist die grafische Benutzeroberfläche des Spiels vorgegeben. Im Konstruktor werden sechs Buttons zur Steuerung des Spiels definiert, die aber noch nicht verwendet werden.

Bearbeiten Sie die Methode **layouten** an der markierten Stelle so, dass diese sechs Buttons dem `JPanel guiRechts` in der Position **BorderLayout.SOUTH** sinnvoll hinzugefügt werden. Sie können dazu beliebige weitere JPanels und Layouts verwenden.

***Hinweis:*** Sie sollen ein sinnvolles GUI-Layout erstellen; Es gibt keine „absolut korrekte“ Lösung!

## Aufgabe 2 Form (1 + 2 = 3 Punkte)

Alle Spielsteine haben eine **Form**. Bearbeiten Sie nun die entsprechende Klasse.

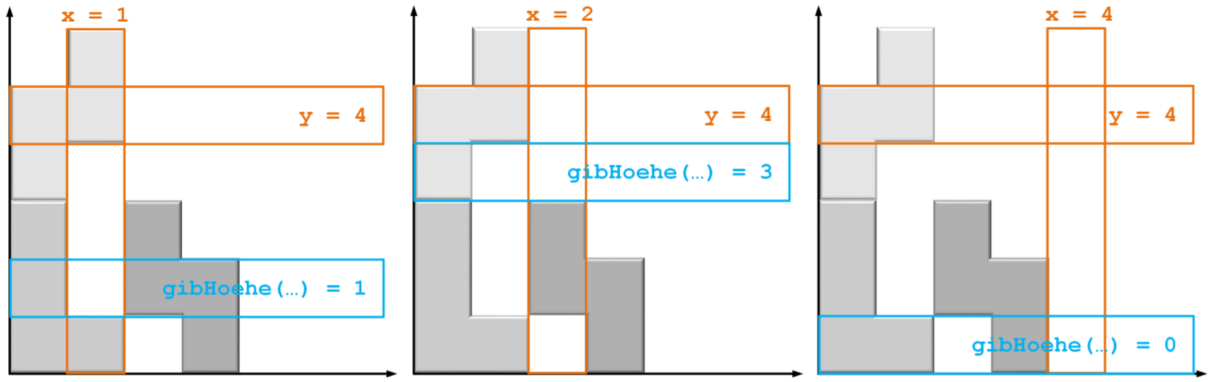
a) Finden Sie das Datenfeld, das bestimmt, welche Blöcke eines rechteckigen Bereichs vom Stein belegt sind und schreiben Sie die Zugriffsmethode `istBelegt(int spalte, int zeile)`. Diese soll genau dann **true** zurückgeben, wenn der angegebene Block belegt ist.

b) Als nächstes wenden Sie sich der Methode **dreheNachLinks()** zu. Diese Methode soll eine um 90° nach links gedrehte **Kopie** einer allgemeinen Form erzeugen. In dieser gedrehten Kopie soll also die linke obere Ecke des Originals an der linken unteren Ecke sein, die linke untere Ecke an der rechten unteren Ecke usw. Beachten Sie, dass Zeilen von unten nach oben nummeriert sind – im fertigen Spiel sollen sich die Spielsteine **gegen den Uhrzeigersinn** drehen.

## Aufgabe 3 Matrix (1 + 2 + 1 + 2 = 6 Punkte)

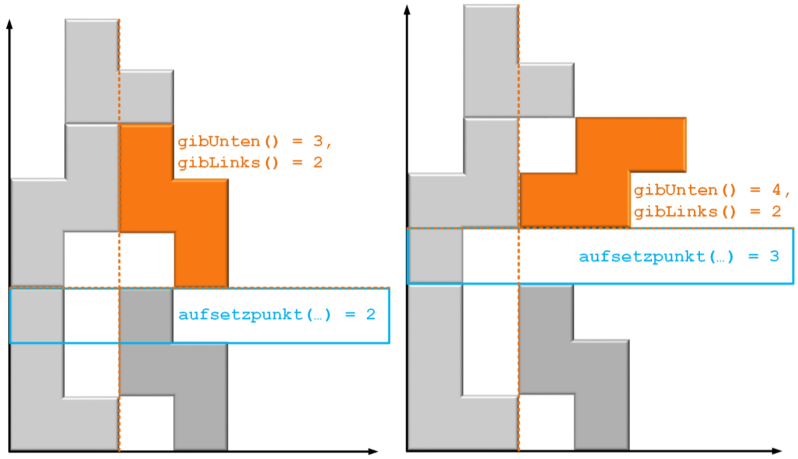
Öffnen Sie nun die Klasse **Matrix**. Hier werden Blöcke in einer zweidimensionalen Datenstruktur **belegung** verwaltet. Eine `ArrayList` enthält die Zeilen der Matrix, die jeweiligen Blöcke dieser Zeile sind in einem Array mit einer festen Länge gespeichert.
In dieser Aufgabe geht es darum, die Matrix mit Steinen zu füllen, was in mehreren Schritten passiert.

a) Sie benötigen zuerst die Methode `gibHoehe(int x, int y)`. Diese soll den Index der Zeile über dem obersten belegten Block in Spalte **x** unterhalb der Zeile **y** zurückgeben. Gibt es keine belegten Blöcke "unter" der übergebenen Koordinate (also in negativer y-Richtung), so soll **0** zurückgegeben werden. Nutzen Sie dabei die Methode `gibBlock(int x, int y)`.




**(a)** `gibHoehe` mit **x**=1, **y**=4 **(b)** `gibHoehe` mit **x**=2, **y**=4 **(c)** `gibHoehe` mit **x**=4, **y**=4

b) Verwenden Sie die vorherige Methode dann, um den Aufsetzpunkt eines Steins zu berechnen. Schreiben Sie dazu die Methode `aufsetzpunkt`, die einen Stein übergeben bekommt und die Zeile zurückgibt, auf der der untere Rand des Stein zum liegen kommt. Beachten Sie dabei, dass ein Stein an seinem unteren Rand nicht komplett besetzt sein muss. Verwenden Sie außerdem die derzeitige legale Position des Steins für Ihre Berechnung. Gehen Sie also davon aus, dass der Stein von seiner jetzigen Position senkrecht nach unten auf die Matrix fällt.




**(a)** `aufsetzpunkt` mit Stein an (2, 3) **(b)** `aufsetzpunkt` mit Stein an (2, 4)

c) Schreiben Sie in der Klasse `Stein` die Methode `fallen`. Diese soll überprüfen, ob die Unterkante des Steins über dem oben berechneten Aufsetzpunkt liegt. Ist dies der Fall, soll die Methode den Stein um einen Block nach unten bewegen und `true` zurückliefern. Ansonsten soll `false` zurückgeliefert werden.

d) Die nächste Methode muss wieder in `Matrix` implementiert werden: Mit der Methode `manifestieren` soll ein Stein in der Matrix auf dem Aufsetzpunkt abgelegt und in Blöcke verwandelt werden. Erzeugen Sie dazu für alle belegten Felder der `Form` des Steins neue Blöcke und fügen diese mittels `setzeBlock` der Matrix hinzu. Verwenden Sie als Stein-Index der neuen Blöcke das Datenfeld `steinIndex` von Stein. Wenn der Aufsetzpunkt kleiner als die maximale Zeilenanzahl der Matrix - 1 ist, soll die Methode `true` zurückgeben, sonst `false`.

## Aufgabe 4 Einfache Physik (2 + 1 = 3 Punkte)

Zunächst soll eine einfache Physik implementiert werden, d.h. alle Blöcke über einer entfernten vollständigen Zeile fallen einfach um eine Zeile nach unten. Wir bleiben dazu in der Klasse `Matrix`, welche das in Aufgabe 3 beschriebene Spielfeld speichert.

a) Schreiben Sie die Methode `faerbeVolleZeilen()`, welche alle Zeilen der Matrix auf Vollständigkeit (:= alle Blöcke dieser Zeile sind *nicht* `null` und *nicht* `schwarz` gefärbt) prüft. Dabei sollen alle vollen Zeilen schwarz gefärbt und das Datenfeld `volleZeilen` entsprechend erhöht werden.

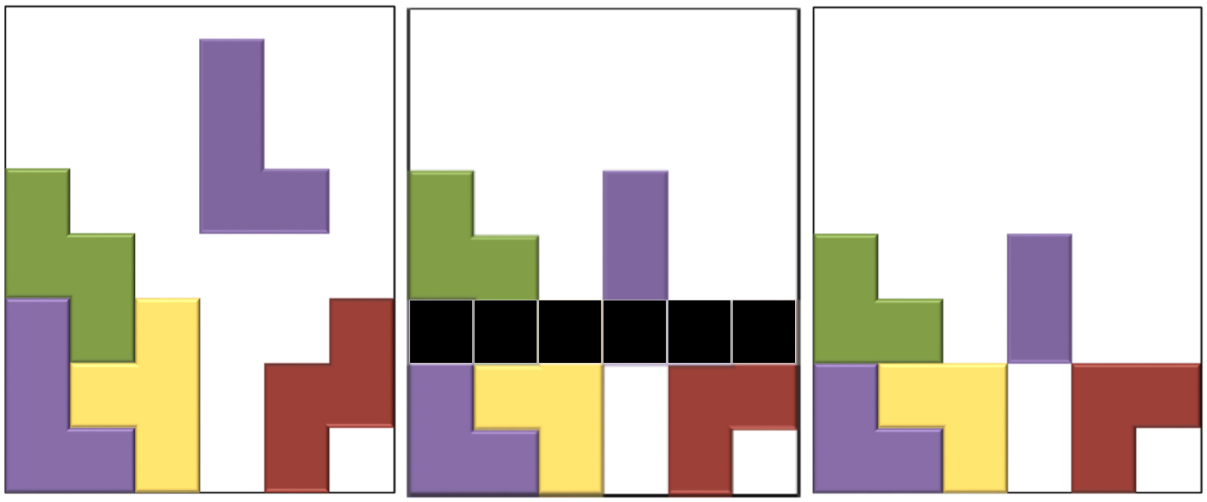
b) Schreiben Sie die Methode `loescheSchwarzeZeilen()`, welche alle schwarzen Zeile der Matrix aus der Liste entfernt. Die Methode soll `true` zurückgeben, wenn mindestens eine Zeile gelöscht wurde.
***Hinweis:*** Sie können z.B. die `remove`-Methode auf einem Iterator verwenden.

Verwendet werden Ihre Methoden beim Ablegen eines Steins in der Methode `ablegen` bzw. beim Aufräumen der Matrix in der Methode `aufraumen`. Durch die Trennung dieser beiden Methoden entsteht ein visuelles Feedback beim Vervollständigen einer Zeile.
In *Abbildung 4* sehen Sie den Effekt dieser einfachen Physik: Der violette Stein fällt herunter und füllt beim Auftreffen auf die Matrix die Lücke in Zeile 3. Daraufhin wird erkannt, dass diese nun voll belegt ist und die Zeile wird gelöscht.

## Aufgabe 5 Erweiterte Physik (2 + 2 + 2 = 6 Punkte)

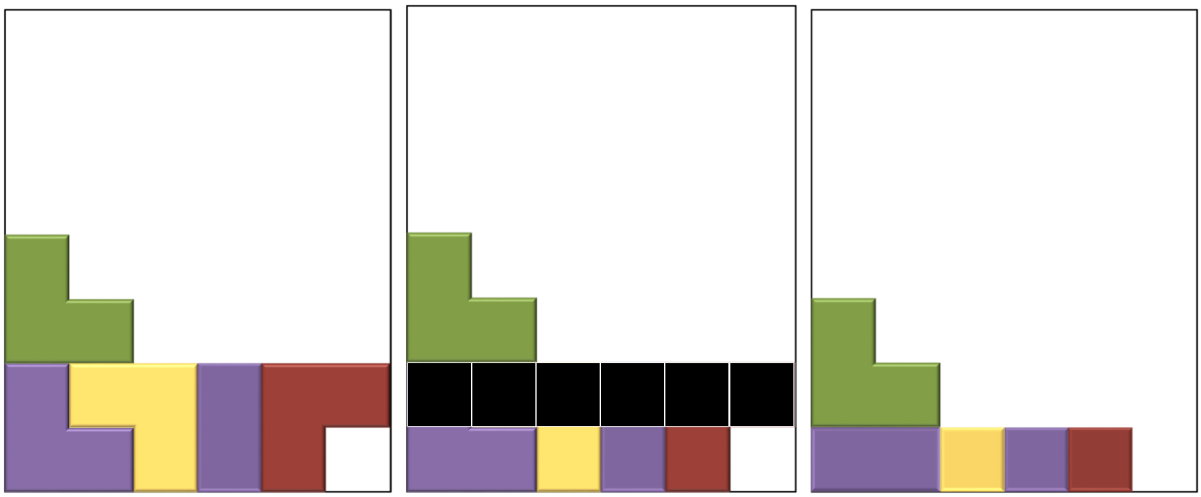
In der vorherigen Aufgabe haben Sie die einfache Physik implementiert, die in den ersten Tetris-Varianten verwendet wurde. Nun soll das Verhalten der Steine etwas realistischer werden. Der Stein, der in *Abbildung 4c)* noch über einer neu geschaffenen Lücke schwebt, soll diese nun auch auffüllen, wie in *Abbildung 5* zu sehen ist. Diese Physik wird auch als „Cascade“ bezeichnet.
Gehen Sie dazu folgendermaßen vor:

a) Zunächst soll ein Stein auf der Matrix markiert werden, also alle zusammenhängenden Blöcke desselben Steins erkannt werden. Verwenden Sie dazu den sogenannten **Floodfill-Algorithmus**, der auch z.B. in Grafikprogrammen zur Anwendung kommt. Dieser Algorithmus ist ausführlich unter `http://de.wikipedia.org/wiki/Floodfill`



**(a)** Ein L-Stein fällt **(b)** Stein trifft auf, die Zeile ist voll **(c)** Die Zeile wird gelöscht

**Abbildung 4:** Ablauf bei einfacher Physik



**(a)** Schwebender Stein fällt herab **(b)** Weitere Zeile ist voll **(c)** Die Zeile wird gelöscht

**Abbildung 5:** Ablauf bei erweiterter Physik

beschrieben. Wir halten uns an die **4-connected**-Variante und verwenden anstelle einer Farbe den Stein-Index der Blöcke. In der Vorgabe ist bereits der Methodenkopf `floodfill(int x, int y, int steinIdx, List<Integer> xs, List<Integer> ys)` angegeben. Wenn der `x`-te Block in Zeile `y` (dafür gibt es bereits eine Methode) existiert und den übergebenen Stein-Index hat, soll dieser aus der Matrix gelöscht werden (auch dafür gibt es schon eine Methode) und seine Koordinaten `x` und `y` in den Listen `xs` bzw. `ys` gespeichert werden. Rufen Sie dann den **Flood-fill**-Algorithmus für die vier benachbarten Blöcke auf, wie im Wikipedia-Artikel zum Algorithmus beschrieben. Sie können davon ausgehen, dass die eingegebenen Listen gleich lang sind.

Jetzt soll der ausgeschnittene Stein auf die Matrix fallen gelassen werden. Das passiert in der Vorgabe bereits in der Methode `aufraeumen`. Ihre Aufgabe besteht darin, aus den Listen der x- und y-Koordinaten der Blöcke einen neuen Stein zu erstellen. Hierfür brauchen Sie eine neue **Form**, und für diese ein zweidimensionales **boolean**-Array. Um die Größe dieses Arrays zu bestimmen, benötigen Sie zunächst die Begrenzungen des ausgeschnittenen Steins.

b) Implementieren Sie die Methode `boundingBox(List<Integer> xs, List<Integer> ys)`. Diese soll die Begrenzungen **links**, **rechts**, **unten** und **oben** (also jeweils den minimalen und maximalen x- bzw. y-Wert) der Listen bestimmen und als Array der Länge 4 in der angegebenen Reihenfolge zurückgeben. Die Methode soll unabhängig von der aktuellen Matrixgröße funktionieren.

c) Implementieren Sie nun die Methode `erstelleStein`, die beide besagten Listen und zusätzlich die Farbe des Steins übergeben bekommt und einen so beschriebenen Stein zurückgibt. Nutzen Sie dabei den Konstruktor `Stein(Matrix matrix, Form form, int unten, int links)`. `matrix` ist `this`; Das Array für form erzeugen Sie mithilfe der `boundingBox`-Methode.

## Aufgabe 6 Formatierung und Kommentierung (*Abzug* bis zu 4 Punkte)

Gut formatierter und kommentierter Programmcode hilft, das Programm verständlicher zu machen – auch dem Korrektor. Wir erwarten von Ihnen, dass Sie ...

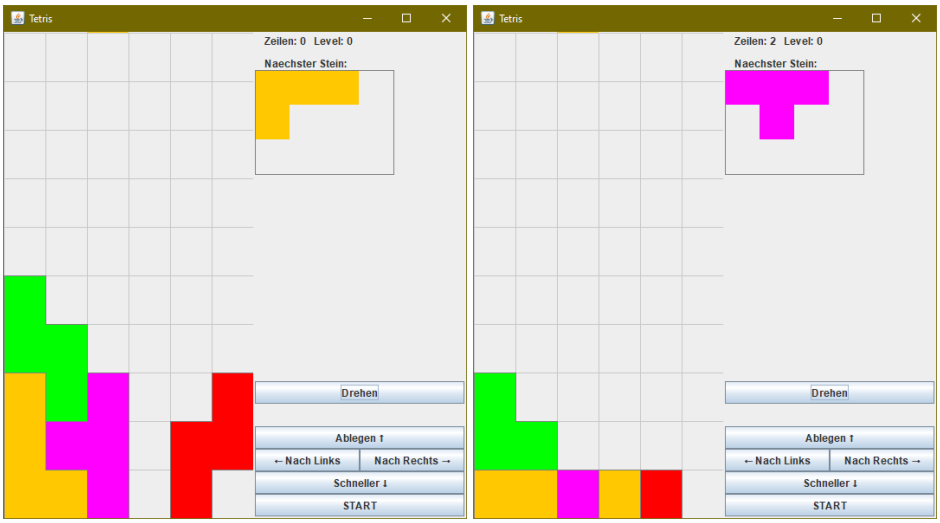
- ... jede von Ihnen bearbeitete Methode ausführlich mit Javadoc-Kommentaren beschreiben (dazu gehören auch `@param` für jeden Parameter und `@return` für Nicht-`void`-Methoden),
- ... die Funktionsweise nicht-trivialer Codeabschnitte mit einfachen Kommentaren (`/* */` oder `//`) beschreiben,
- ... sich an den Standard-Einrückungsstil halten, also untergeordnete Blöcke (wie z. B. Methoden, Schleifen, bedingte Anweisungen) mit Tabulator oder Leerzeichen einrücken,
- ... sinnvolle Variablennamen verwenden.

Sollten Sie diese Regeln nicht einhalten, werden wir bei der Bewertung bis zu `vier Punkte` abziehen.

***Hinweis:*** Denken Sie ggf. auch an die von Ihnen erstellten privaten Hilfsmethoden!
Sie können deutsch oder englisch kommentieren – entscheiden Sie sich für eine Variante!

## Aufgabe 7 Testen und Starten des Spiels

Testen Sie Ihre Methoden gründlich – es reicht nicht aus, das Spiel laufen zu lassen! Sie haben in der AI1 noch viele andere Möglichkeiten kennen gelernt, beispielsweise „manuelles Ausführen“, Testmethoden, Unit-Tests, Bildschirmausgaben...
Um das Spiel zu starten kompilieren Sie alle Quellcode-Dateien in den Unterpaketen und starten die `main`-Methode der Klasse `Launch`. Das Spiel kann durch die von Ihnen in Aufgabe 1 erstellte GUI oder über die Tastatur gesteuert werden. Die vier Pfeiltasten entsprechen den vier Pfeilbuttons (siehe Vorstellung ProPro), der aktuelle Stein wird mit der Taste „A“ gedreht. Das Spiel startet erst, wenn Sie auf den Start-Button klicken.
Wir stellen Ihnen außerdem die Klasse `Demo` zur Verfügung, die automatisch ein Demozenario erstellt. Erzeugen Sie dazu eine Instanz der Klasse und führen Sie nacheinander die Methoden `start` und `fuellen` aus. Wenn Sie die Aufgaben korrekt bearbeitet haben, sollte Ihre Zeichenfläche den Screenshots aus folgender Abbildung entsprechen. Die Anordnung der Buttons kann sich jedoch von Ihrer Version unterscheiden!




**(a)** Zeichenfläche nach `start` **(b)** Zeichenfläche nach `fuellen`

## Wir wünschen Ihnen viel Spaß und Erfolg!