Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE26-OOSoftwareentwicklung
(Stand 23.11.2011)

Aufgabe 1:

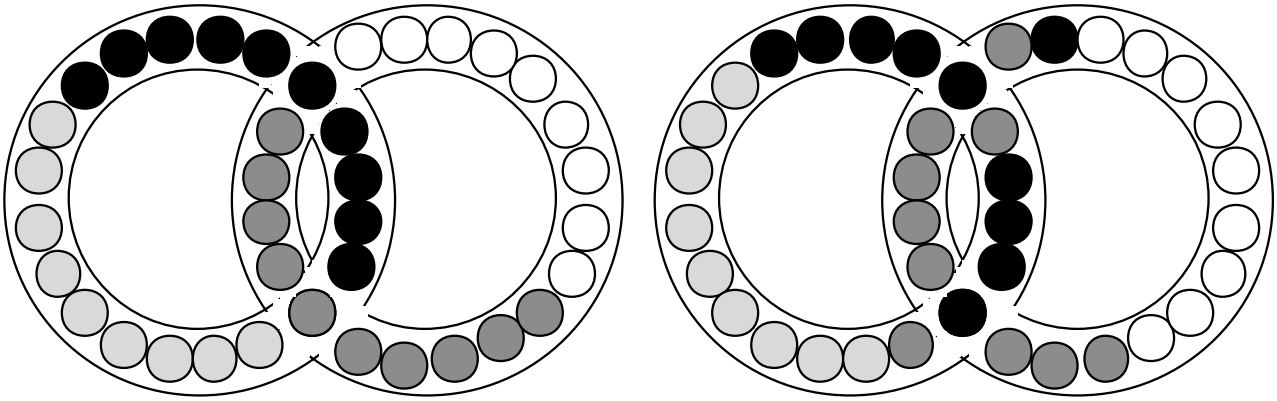
Rubik`s Ringe enthalten 10 schwarze, 9 gelbe, 10 rote und 9 graue Kugeln. Die Kugeln eines Ringes können linksdrehend oder rechtsdrehend verschoben werden. Man kann jeden Ring einzeln drehen. Durch die beiden Schnittpositionen können Kugeln von einem Ring in den anderen übergehen.

Beispiel:

Der Zustand in der Abbildung rechts ist aus dem Zustand links entstanden durch:

1. Drehung des rechten Ringes um zwei Kugeln nach rechts (im Uhrzeigersinn),
2. Drehung des linken Ringes um eine Kugel ebenfalls nach rechts.

Ziel des Spiels ist, durch Drehen der Ringe aus einer bestimmten Anordnung der Kugeln (Anfangszustand) eine andere Anordnung der Kugeln (Endzustand) herzustellen.



Schreiben Sie ein Java-Programm, das zunächst einen Anfangszustand einliest. Anschließend werden (in einer Endlosschleife) jeweils ein Spielzug eingelesen und der erreichte Zustand auf dem Bildschirm dargestellt (kreisförmige Darstellung der Ringe ist **nicht** erforderlich!). Eine Überprüfung der aktuellen Anordnung der Kugeln mit einem Endzustand ist **nicht** erforderlich!

Sie sollen das Rubik-Spiel auf eine objektorientierte Art und Weise implementieren. Entwerfen Sie daher geeignete Klassen **RubikSpiel**, **Ring**, **Kugel** und **Spielzug!**

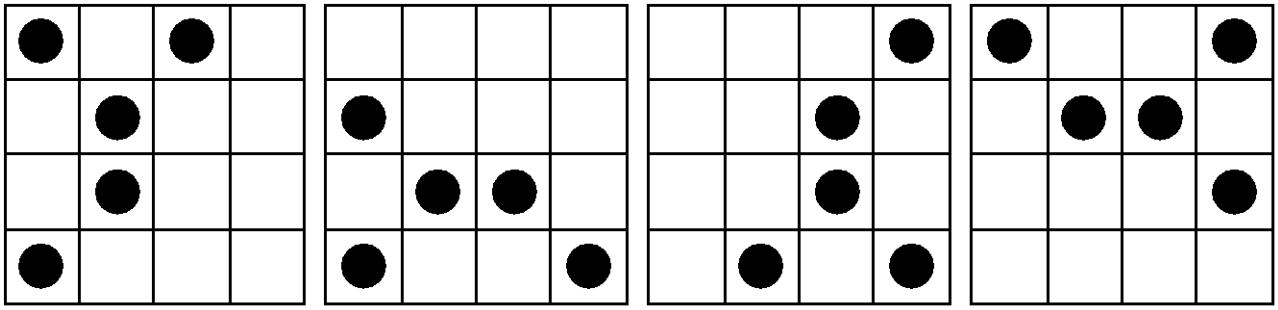
Mögliche Vorgehensweise:

1. Entwerfen Sie zunächst die Klassen **Ring** und **Kugel**. Überlegen Sie sich dazu eine geeignete Datenstruktur zur Repräsentation der Ringe. Bedenken Sie dabei: Es gibt zwei kreisförmig zu organisierende Mengen (Ringe) mit jeweils 20 Kugeln, wobei sich die beiden Ringe jedoch zwei Kugeln teilen.
2. Entwerfen Sie dann die Klasse **Spielzug**. Überlegen Sie sich dazu eine geeignete Datenstruktur zur Repräsentation von Spielzügen.
3. Implementieren Sie die Ein- und Ausgabemethoden.
4. Implementieren Sie den Drehungsalgorithmus.
5. Implementieren Sie das Rahmenprogramm (Klasse **RubikSpiel**).

Aufgabe 2:

Puan-Puan ist ein Spiel, das von zwei Personen auf einem Spielbrett von n mal n Feldern (n = 2, 3, 4, ..., 9) gespielt wird. Als Spielfiguren stehen n mal n gleichfarbige Steine zur Verfügung. Die Spieler ziehen abwechselnd. Bei einem Zug wird entweder ein Stein gesetzt oder ein bereits gesetzter Stein weggenommen.

Wer eine Steinstellung erzeugt, die schon einmal im Verlauf des aktuellen Spiels aufgetreten ist oder die bis auf eine Drehung des gesamten Spielfeldes um 90, 180 oder 270 Grad mit einer bereits aufgetretenen Stellung übereinstimmt, verliert das Spiel. Das Spiel endet, wenn einer der beiden Spieler verliert bzw. nach 100 Zügen. In letzterem Falle endet das Spiel mit einem Unentschieden.



Vier (bis auf Drehung) gleiche Stellungen („Puan-Gleichheit“)

Schreiben Sie ein Java-Programm, das zuerst die Größe des Spielbrettes einliest und dann abwechselnd die Spielzüge beider Spieler entgegennimmt, bis das Spiel endet

Aufgabe 3:

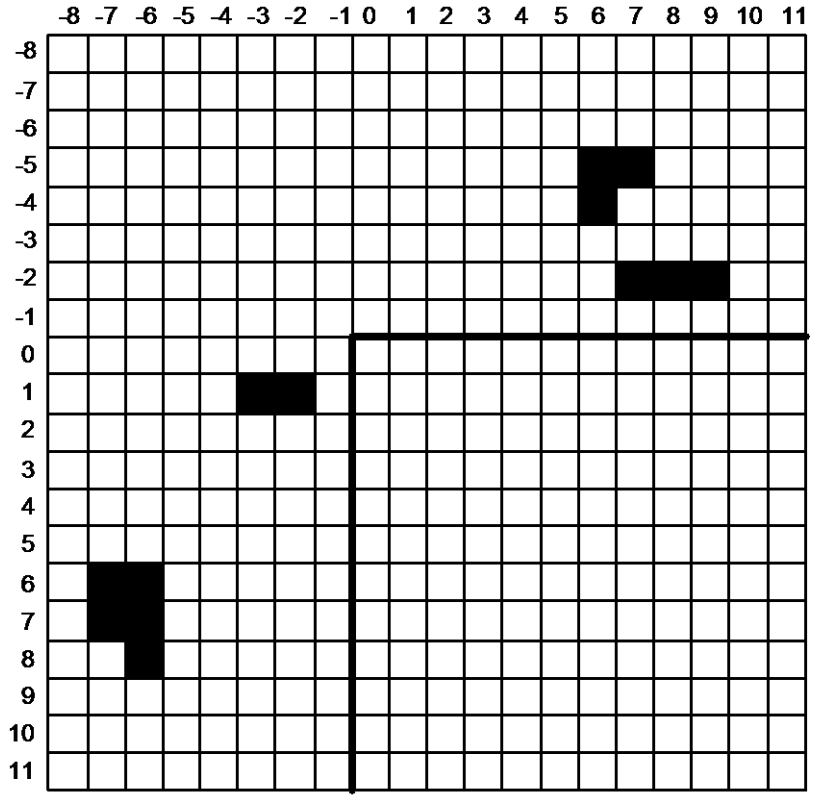
Quadratien ist ein quadratisches Gebiet aus quadratischen Feldern. Das Feld in der Nordwest-Ecke hat die Zeilennummer 0 und die Spaltennummer 0.

Das Wetter in Quadratien wird durch quadratische Wolken bestimmt, die genau ein Feld groß sind. Solche Wolken rücken getaktet über Quadratien vor, und zwar von Norden nach Süden 2 Felder pro Takt und - in einer anderen Höhe - von Westen nach Osten 3 Felder pro Takt. Es regnet überall dort, wo sich nach einem Vorrücken sowohl eine Nord-Süd- als auch eine West-Ost-Wolke befindet. Wolken, aus denen es regnet, lösen sich auf.

Die Wolkenvorhersage gibt an, an welchen Stellen (in der Form: Zeilennummer/Spaltennummer) sich zum aktuellen Zeitpunkt Wolken befinden. Daraus läßt sich dann ermitteln, wo es in Quadratien regnen wird; denn es werden nur solche Wolken angegeben, die über Quadratien hinwegziehen werden.

Beispiel: Wolkenvorhersage:

```
-5/6 -4/6 -2/7 7/-6 -2/9 -5/7 1/-2 8/-6 6/-6 7/-7 6/-7 1/-3 -2/8
```




```
Es wird an folgenden Stellen regnen: 1/6, 1/7, 8/9
```

Aufgabe: Entwickeln Sie ein Java-Programm, welches folgendes leistet:

1. Einlesen der Größe von Quadratien (Anzahl Zeilen bzw. Spalten).

2. Einlesen einer Wolkenvorhersage.
3. Ausgabe, wo in Quadratien Regen fällt.

Besondere Anforderungen: Bei der Programmentwicklung sind folgende Anforderungen zu berücksichtigen:

- Wolken dürfen in der Wettervorhersage nur links bzw. oberhalb von Quadratien erzeugt werden. Unter Einhaltung dieser Restriktion dürfen (prinzipiell) beliebig positionierte Wolken eingegeben werden. Eine Beschränkung der Größe des Umlandes von Quadratien ist nicht erlaubt.
- Die Menge der Wolken in der Wettervorhersage ist (prinzipiell) beliebig groß. Es darf vorweg keine Größenbeschränkung angegeben werden.
- Entwickeln Sie eine Klasse Wolke, die Wolken realisiert. Überlegen Sie sich notwendige Attribute (Zeile, Spalte, ...) und Methoden (Überprüfung auf gleichePosition, vorrücken, ...) der Klasse und implementieren Sie diese.
- Entwickeln Sie eine Klasse Wolkenmenge, die eine prinzipiell beliebig große Menge von Wolken verwalten kann und auf der Wettersimulationen möglich sind. Überlegen Sie sich notwendige Attribute (Realisierung als verkettete Liste, ...) und Methoden (Einfügen von Wolken, Entfernen von Wolken, Wettersimulation, ...) der Klasse und implementieren Sie diese.
- Entwickeln Sie das Hauptprogramm. Sie können dabei bspw. folgendermaßen vorgehen: Erzeugen Sie ein Wolkenmenge-Objekt. Erzeugen Sie während des Einlesens der Wolkenvorhersage jeweils ein neues Wolke-Objekt für jede Wolke und fügen Sie dieses in das Wolkenmenge-Objekt ein. Lassen Sie nach Beendigung der Wolkenvorhersage auf dem Wolkenmenge-Objekt eine Wettersimulation ausführen bis keine Wolken mehr da sind (abgeregnet oder über Quadratien hinweggetrieben). Speichern Sie während der Simulation die Regenwolken in einem zweiten Wolkenmenge-Objekt ab. Geben Sie nach der Simulation die Wolke-Objekte, die sich im zweiten Wolkenmenge-Objekt befinden, auf den Bildschirm aus**.**

Beispiel für die Benutzerschnittstelle des Programms

```
> java Quadratien
Geben Sie bitte die Größe von Quadratien ein:
> 12
Geben Sie bitte die Wettervorhersage ein:
weitere Wolke (y/n)
> y
Zeile angeben:
> 0
Spalte angeben:
> -1
weitere Wolke (y/n)
> y
Zeile angeben:
> -2
Spalte angeben:
> 2
weitere Wolke (y/n)
> n
Es wird an folgenden Stellen regnen: 0/2.
```

Aufgabe 4:

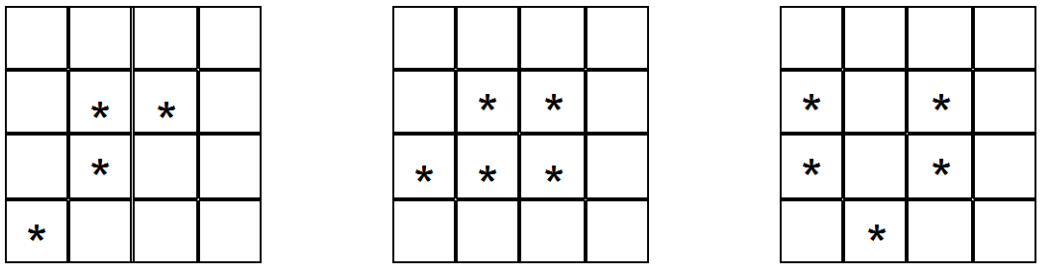
Das „Game-of-Life“ wird auf einem schachbrettartigen Feld gespielt, das eine „Bevölkerung“ von „toten“ und „lebenden“ Zellen darstellt. Jede Zelle kann „überleben“, „sterben“ oder „geboren“ werden. Die schrittweise Entwicklung von einem Stellungsbild zum nächsten erfolgt gemäß einiger Regeln, die berücksichtigen, wie viele lebende Nachbarzellen eine Zelle hat. Eine Zelle x, die nicht am Spielfeldrand liegt, hat 8 Nachbarzellen, Zellen am Spielfeldrand entsprechend weniger.

Die Regeln, nach denen sich die Population von einer Stellung zur nächsten entwickelt, sind:

1. Für eine Zelle x, die gerade tot ist, gilt: Wenn x genau 3 lebende Nachbarzellen hat, wird x neu geboren; sonst bleibt x tot.
2. Für eine Zelle x, die gerade lebendig ist, gilt: Wenn x weniger als 2 lebende Nachbarn hat, stirbt x an Vereinsamung; wenn x 2 oder 3 lebende Nachbarzellen hat, bleibt x in der nächsten Stellung lebendig. In allen anderen Fällen stirbt x an Überbevölkerung.

Alle Veränderungen gemäß dieser Regeln geschehen gleichzeitig. Die Simulation beginnt mit einer bestimmten eingelesenen Verteilung von lebenden und toten Zellen. Sie endet nach einer bestimmten eingelesenen Anzahl von Entwicklungsschritten, jedoch früher, wenn sich die Population nicht mehr ändert. Das Spielfeld sollte aus mindestens 15x15 Feldern bestehen.

Beispiel:



Population 1 Population 2 Population 3

Aufgabe: Implementieren Sie das „Game-of-Life“ mit objektorientierten Mitteln.

Aufgabe 5:

In dieser Aufgabe sollen Sie Umfragen simulieren. Zum Hintergrund der Simulation: Eine Umfrage besteht aus mehreren Fragen. Bei den Fragen soll es sich ausschließlich um Fragen handeln, die mit Ja oder Nein beantwortet werden können. An einer Umfrage können beliebig viele Personen teilnehmen. Sie bekommen die Fragen gestellt und müssen mit Ja oder Nein antworten.

Die Simulation soll folgendermaßen ablaufen:

- Zunächst werden die einzelnen Fragen eingegeben.
- Anschließend wird die Umfrage durchgeführt, d.h. mehreren Umfrageteilnehmern werden die einzelnen Fragen gestellt, die diese beantworten müssen.
- Abschließend werden die Umfrageergebnisse auf den Bildschirm ausgegeben. Konkret wird für jede Frage die absolute und prozentuale Anzahl an Ja- und Nein-Antworten ausgegeben.

Aufgabe: Führen Sie, wie in der Unterrichtseinheit beschrieben, eine objektorientierte Entwicklung eines entsprechenden Java-Programms durch. Überlegen Sie zunächst: Was für Objekte bzw. Klassen lassen sich identifizieren, welche Beziehungen existieren zwischen den Objekten, was für Eigenschaften und Funktionen besitzen die Objekte. Implementieren Sie anschließend die Klassen sowie die eigentliche Simulation.

Ablaufbeispiel:

```
Fragen eingeben
----------------
Titel der Umfrage: PK-Java-Umfrage
Anzahl Fragen: 2
Frage 1: Sind die Vorlesungen verstaendlich?
Frage 2: Sind die Uebungsaufgaben zu schwer?

Umfrage
--------
Weiterer Teilnehmer (j/n)?j
Sind die Vorlesungen verstaendlich?
ja/nein (j/n)?j
Sind die Uebungsaufgaben zu schwer?
ja/nein (j/n)?n
Weiterer Teilnehmer (j/n)?j
Sind die Vorlesungen verstaendlich?
ja/nein (j/n)?j
Sind die Uebungsaufgaben zu schwer?
ja/nein (j/n)?n
Weiterer Teilnehmer (j/n)?j
Sind die Vorlesungen verstaendlich?
ja/nein (j/n)?n
Sind die Uebungsaufgaben zu schwer?
ja/nein (j/n)?n
Weiterer Teilnehmer (j/n)?n

Umfrageergebnisse
-----------------
Umfrage: PK-Java-Umfrage
Frage: Sind die Vorlesungen verstaendlich?
ja-Antworten: 2 = 66.66666666666667 Prozent
nein-Antworten: 1 = 33.333333333333336 Prozent
Frage: Sind die Uebungsaufgaben zu schwer?
ja-Antworten: 0 = 0.0 Prozent
nein-Antworten: 3 = 100.0 Prozent
```

Aufgabe 6:

In dieser Aufgabe sollen Sie EMail-Verkehr simulieren. Zum Hintergrund der Simulation: Personen können sich gegenseitig EMails zuschicken. Jede Person hat einen Namen, eine EMail-Adresse und jeder Person ist eine Mailbox zugeordnet, in der empfangene EMails gespeichert werden. Jede EMail besitzt neben dem Sender und Empfänger (nur einer, keine Gruppen!) ein Subject und den eigentlichen Inhalt.

Die Simulation soll folgendermaßen ablaufen:

- Zunächst melden sich eine Menge von Personen mit Name und EMail-Adresse an.
- Anschließend wird der EMail-Verkehr simuliert, d.h. es werden eine Menge von EMails geschrieben und verschickt. Dabei gilt: Bekannt sind nur jeweils die Namen der Personen, an die eine EMail verschickt werden soll, nicht die EMail-Adresse selbst!
- Abschließend soll es (in einer Schleife) möglich sein, für eine anzugebende Person (Name) die erhaltenen EMails auf den Bildschirm auszugeben.

Aufgabe: Führen Sie, wie in der Vorlesung beschrieben, eine objektorientierte Entwicklung eines entsprechenden Java-Programms durch. Überlegen Sie zunächst: Was für Objekte bzw. Klassen lassen sich identifizieren, welche Beziehungen existieren zwischen den Objekten, was für Eigenschaften und Funktionen besitzen die Objekte. Implementieren Sie anschließend die Klassen sowie die eigentliche Simulation.

Ablaufbeispiel:

```
Anmeldung
---------
Personenanzahl: 2
Name: dibo
EMail: boles@informatik.uni-oldenburg.de
Name: Klaus
EMail: klaus@uni-oldenburg.de

EMailverkehr
------------
Weitere EMail verschicken (j/n)?j
Sendername: dibo
Empfaengername: Klaus
Subject: Hallo
Text: nur ein Gruss von dibo
Weitere EMail verschicken (j/n)?j
Sendername: dibo
Empfaengername: Klaus
Subject: Nochmal hallo
Text: ein weiterer Gruss von dibo
Weitere EMail verschicken (j/n)?n
```

```
Mailboxabfrage
--------------
Weitere Mailboxabfrage (j/n)?j
Name: dibo
Mailbox ist leer
Weitere Mailboxabfrage (j/n)?j
Name: Klaus
****** Begin EMail *****
TO: klaus@uni-oldenburg.de(Klaus)
FROM: boles@informatik.uni-oldenburg.de(dibo)
SUBJECT: Hallo

nur ein Gruss von dibo
****** End EMail *****
****** Begin EMail *****
TO: klaus@uni-oldenburg.de(Klaus)
FROM: boles@informatik.uni-oldenburg.de(dibo)
SUBJECT: Nochmal hallo

ein weiterer Gruss von dibo
****** End EMail *****
Weitere Mailboxabfrage (j/n)?n
```

Aufgabe 7: Opala

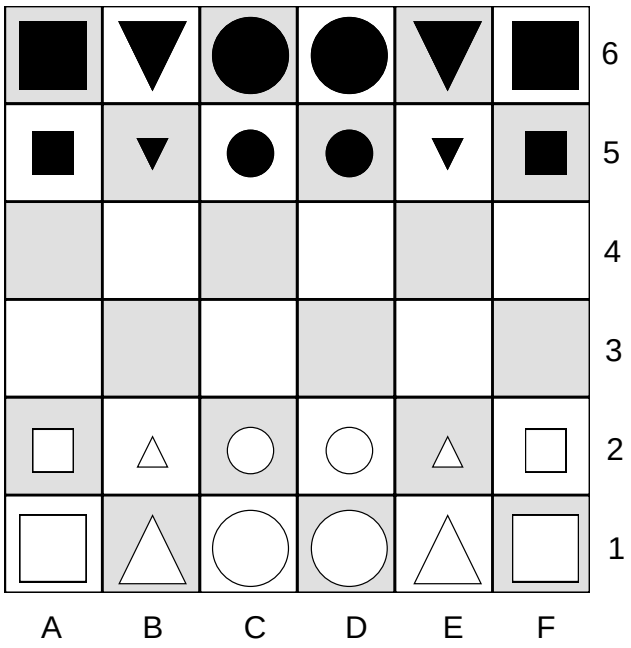
Opala - Spielregeln

Spielbrett

Gespielt wird auf einem Schachbrett mit allerdings nur 6x6 Feldern.

Spielfiguren

Spielfiguren sind geometrische Gebilde. Es gibt große und kleine Quadrate, große und kleine Kreise sowie große und kleine Dreiecke. In der Abbildung ist die Anfangs-aufstellung zu sehen.



Spieler

OPALA ist ein Spiel für zwei Spieler. Spieler A (Weiß) spielt mit weißen Spielfiguren von unten nach oben. Spieler B (Schwarz) spielt mit schwarzen Spielfiguren von oben nach unten.

Spielablauf

Es wird immer abwechselnd gezogen, wobei Spieler Weiß beginnt. Es besteht Zugzwang.

Spielzüge

Für die einzelnen Spielfiguren gelten folgende Zugregeln:

- Große Quadrate dürfen wie Türme beim Schach gezogen werden, d.h. beliebig (>0) viele Felder horizontal oder vertikal.
- Kleine Quadrate dürfen wie große Quadrate gezogen werden, allerdings nur ein einzelnes Feld.
- Große Dreiecke dürfen wie Läufer beim Schach gezogen werden, d.h. beliebig (>0) viele Felder in einer Diagonalen.
- Kleine Dreiecke dürfen wie große Dreiecke gezogen werden, allerdings nur ein einzelnes Feld.
- Große Kreise dürfen wie Damen beim Schach gezogen werden, d.h. beliebig (>0) viele Felder in einer Horizontalen, Vertikalen oder Diagonalen.
- Kleine Kreise dürfen wie große Kreise gezogen werden, allerdings nur ein einzelnes Feld

Dabei gelten grundsätzlich folgende Resultate bzw. Einschränkungen:

- Bei einem Spielzug darf nur eine einzelne eigene Spielfigur gezogen werden.
- Bei einem Spielzug dürfen keine Spielfiguren übersprungen werden.
- Es darf nicht auf ein Feld gezogen werden, auf dem bereits eine eigene Spielfigur steht.
- Wird eine Spielfigur auf ein Feld gezogen, auf dem eine Spielfigur des Gegners steht, so wird diese geschlagen, d.h. vom Spielbrett entfernt.

Ziel des Spiels

Ziel des Spiel ist es, mit einer eigenen Spielfigur die gegenüber liegende Grundlinie zu erreichen, und zwar ohne dass diese im nächsten Zug wieder geschlagen werden könnte.

Ende des Spiels und Sieger

Ein Spiel ist beendet

- wenn ein Spieler X mit einer eigenen Spielfigur die gegenüber liegende Grundlinie erreicht hat und diese Spielfigur im nächsten Zug nicht direkt geschlagen werden kann. In diesem Fall hat Spieler X unmittelbar gewonnen.
- wenn ein Spieler X mit einer eigenen Spielfigur die gegenüber liegende Grundlinie erreicht hat und diese Spielfigur zwar im nächsten Zug direkt geschlagen werden kann, sein Gegner dies jedoch nicht tut. In diesem Fall hat Spieler X gewonnen, nachdem der Gegner seinen Zug ausgeführt hat.
- wenn ein Spieler, wenn er am Zug ist, keinen legalen Spielzug mehr ausführen kann. In diesem Fall hat sein Gegner gewonnen.
- sobald ein Spieler keine eigenen Spielfiguren mehr besitzt. In diesem Fall hat der Gegner gewonnen.

Aufgabe: Implementieren Sie mit objektorientierten Mitteln das Spiel Opala, so dass es 2 Menschen gegeneinander spielen können.

Aufgabe 8: BreakThrough

Bei dieser Aufgabe sollen Sie ein Programm entwickeln, durch das zwei menschliche Spieler am Computer gegeneinander ein kleines Spiel namens *BreakThrough* spielen können.

Regeln

BreakThrough ist ein Spiel, das von zwei Personen auf einem Spielbrett von n Reihen und m Spalten (n und m = 5, 6, ... oder 9) gespielt wird.

Die Spieler (A und B) haben jeweils 2 * m gleichartige Figuren in ihrer Spielerfarbe (auf dem Brett). Sie werden zu Beginn auf den beiden Reihen aufgestellt, die dem jeweiligen Spieler am nächsten sind (wie bei der Schach-Grundstellung).

Die Spieler ziehen abwechselnd, A beginnt. Wer am Zug ist, muss ziehen, Passen ist nicht erlaubt.

Ein Zug wird gemacht, indem man eine seiner Figuren auf das Feld direkt vor der Figur zieht oder auf ein Feld diagonal vor der Figur. Das Feld, auf das gezogen wird, muss leer oder vom Gegner besetzt sein. Im letzten Fall wird die gegnerische Figur durch den Zug geschlagen, d. h. vom Brett genommen. Ziehen und Schlagen geht immer nur ein Feld weit, man kann keine Felder überspringen.

Man gewinnt, indem man eine seiner Figuren auf die hinterste Reihe auf der gegnerischen Seite bringt, oder indem man alle gegnerischen Figuren schlägt.

Aufgabe

Schreiben Sie ein objektorientiertes Java-Programm, mit dessen Hilfe zwei menschliche Spieler das Spiel BreakThrough spielen können.

Hinweise:

Der generelle Spielablauf lässt sich folgendermaßen skizzieren:

- Aufbau des Ausgangsspielbretts
- Ausgabe des Spielbretts
- Spieler A beginnt
- Solange das Spiel nicht beendet ist, tue folgendes
    - Spielzug einlesen
    - Spielzug überprüfen
    - Falls Spielzug nicht korrekt ist, Spiel beenden und Sieger verkünden
    - Falls Spielzug korrekt ist,
        - Spielzug ausführen
        - Ausgabe des Spielbretts
        - Spielerwechsel
- Sieger verkünden

Beispiel für einen Programmablauf mit n = 7 und m = 8 (Benutzereingaben stehen in Klammern (<>)):

```
  Spieler A
  0 1 2 3 4 5 6 7
0 + + + + + + + +
1 + + + + + + + +
2 . . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 o o o o o o o o
6 o o o o o o o o
  Spieler B

Spieler A ist am Zug!
von Reihe: 1
von Spalte: 0
nach Reihe: 2
nach Spalte: 0

  Spieler A
  0 1 2 3 4 5 6 7
0 + + + + + + + +
1 . + + + + + + +
2 + . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 o o o o o o o o
6 o o o o o o o o
  Spieler B

Spieler B ist am Zug!
von Reihe: 5
von Spalte: 0
```

```
nach Reihe: 4
nach Spalte: 1

  Spieler A
  0 1 2 3 4 5 6 7
0 + + + + + + + +
1 . + + + + + + +
2 + . . . . . . .
3 . . . . . . . .
4 . o . . . . . .
5 . o o o o o o o
6 o o o o o o o o
  Spieler B

Spieler A ist am Zug!
von Reihe: 2
von Spalte: 0
nach Reihe: 3
nach Spalte: 1

  Spieler A
  0 1 2 3 4 5 6 7
0 + + + + + + + +
1 . + + + + + + +
2 . . . . . . . .
3 . + . . . . . .
4 . o . . . . . .
5 . o o o o o o o
6 o o o o o o o o
  Spieler B

Spieler B ist am Zug!
von Reihe: 4
von Spalte: 1
nach Reihe: 3
nach Spalte: 1

  Spieler A
  0 1 2 3 4 5 6 7
0 + + + + + + + +
1 . + + + + + + +
2 . . . . . . . .
3 . o . . . . . .
4 . . . . . . . .
5 . o o o o o o o
6 o o o o o o o o
  Spieler B

...
```

Aufgabe 9: Smess

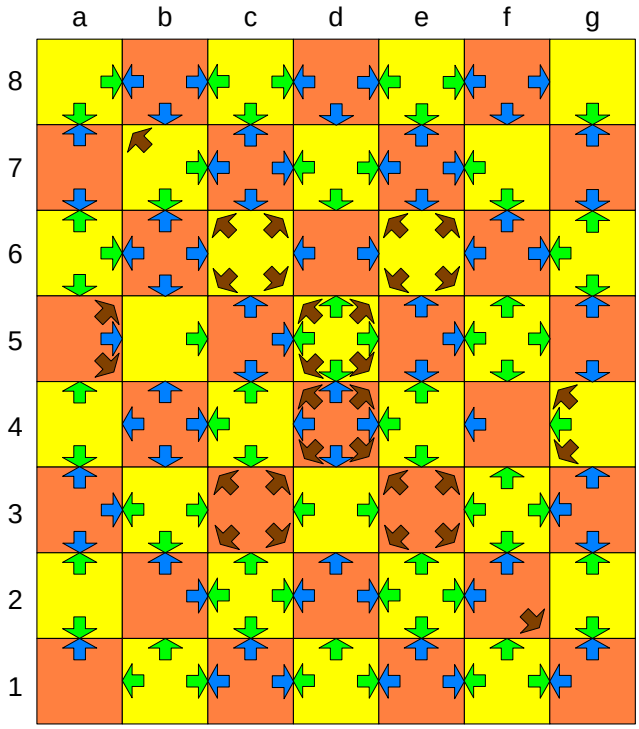
Bei dieser Aufgabe geht es um die Implementierung des Zwei-Personen-Strategiespiels *Smess*. Zunächst die Regeln:

Spieler:

Smess ist ein Spiel für zwei Spieler: Spieler A und Spieler B.

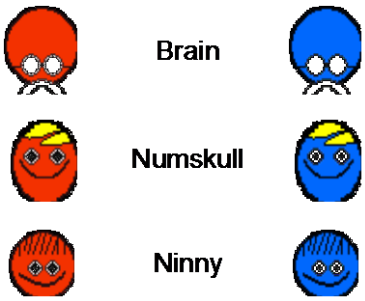
Spielbrett:

Smess wird auf einem Schachbrett-artigen Brett mit 8 x 7 Feldern gespielt (8 Reihen, 7 Spalten). Auf jedem Feld befinden sich Symbole, die Richtungen andeuten. Um einzelne Felder identifizieren zu können, werden Ihnen Koordinaten zugeordnet; die Reihen werden von unten nach oben von 1 bis 8 durchnummeriert, die Spalten von links nach rechts von a bis g.



Spielfiguren:

Spieler A besitzt 12 blaue Spielfiguren, Spieler B 12 rote Spielfiguren. Jeder Spieler besitzt dabei 1 „Brain“, 4 „Numskulls und 7 „Ninnys“.

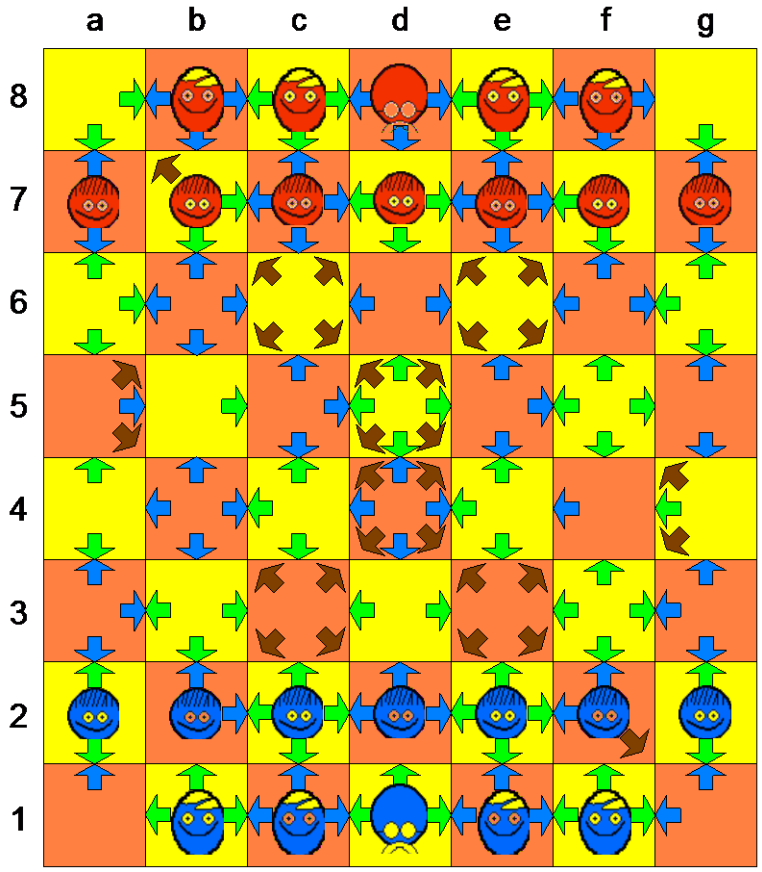


Startaufstellung:

Die blauen Spielfiguren von Spieler A werden in den Reihen 1 und 2 platziert, die roten Spielfiguren von Spieler B in den Reihen 7 und 8. Im Einzelnen werden die Figuren auf folgenden Feldern platziert:

- Blaues Brain: 1d
- Blaue Numskulls: 1b, 1c, 1e, 1f
- Blaue Ninnys: 2a, 2b, 2c, 2d, 2e, 2f, 2g

- Rotes Brain: 8d
- Rote Numskulls: 8b, 8c, 8e, 8f
- Rote Ninnys: 7a, 7b, 7c, 7d, 7e, 7f, 7g



Spielzüge:

Ein Spielzug besteht aus dem Verschieben einer Spielfigur auf dem Spielbrett. Jeder Spieler kann dabei nur seine eigenen Spielfiguren ziehen. Dabei gelten folgende Zugregeln:

- **Ninny**: Darf in einem Spielzug auf ein Nachbarfeld verschoben werden, und zwar in eine der Richtungen, die auf dem Feld angezeigt wird, auf der die Figur gerade steht.
- **Numskull**: Darf eine beliebige Anzahl an Felder verschoben werden, und zwar in eine der Richtungen, die auf dem Feld angezeigt wird, auf der die Figur gerade steht. Dabei dürfen keine anderen Spielfiguren übersprungen werden (auch keine eigenen).
- **Brain**: Darf in einem Spielzug auf ein Nachbarfeld verschoben werden, und zwar in eine der Richtungen, die auf dem Feld angezeigt wird, auf der die Figur gerade steht.

Dabei gilt:

- Spielfiguren dürfen nicht auf Felder verschoben werden, auf denen bereits andere eigene Figuren stehen.

- Wird eine Spielfigur auf ein Feld verschoben, auf der eine Spielfigur des Gegners steht, so wird diese geschlagen, d.h. vom Spielbrett entfernt.
- Wird ein Ninny auf ein Startfeld eines Numskulls des Gegners gezogen, wird das Ninny zu einem Numskull umgewandelt.

Spielablauf:

Es wird abwechselnd gezogen. Spieler A beginnt. Das Spiel ist beendet,

- wenn ein Spieler am Zug ist und nicht ziehen kann oder
- wenn ein Brain geschlagen wird oder
- wenn sich nur noch die beiden Brains auf dem Spielbrett befinden.

Sieger, Verlierer:

Ist ein Spieler am Zug und kann nicht ziehen, so hat er verloren. Wird das Brain eines Spielers geschlagen, so hat er verloren. In beiden Fällen ist der andere Spieler Sieger. Ein Spiel endet Unentschieden, wenn sich nur noch die beiden Brains auf dem Spielbrett befinden.

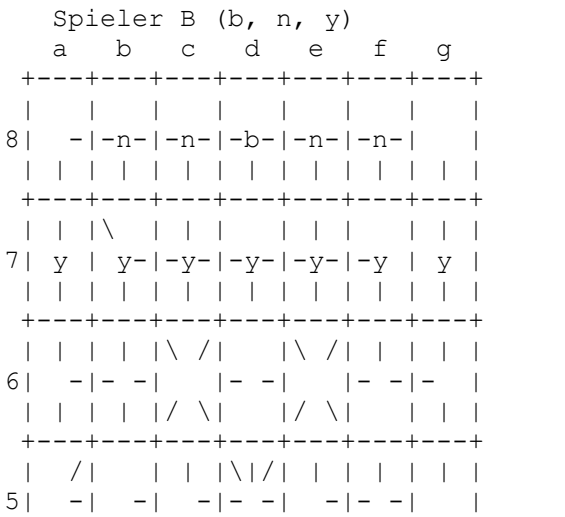
Online:

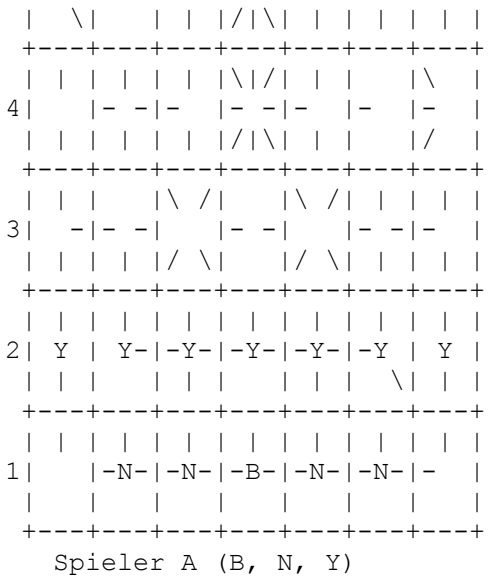
Sie können das Spiel über folgenden URL online spielen:

http://www-is.informatik.uni-oldenburg.de/~dibo/teaching/pkjava/objekttheater/plays/smess/performance.html

Aufgabe: Implementieren Sie das *Smess*-Spiel, so dass zwei Menschen gegeneinander *Smess* spielen können! Gehen Sie bei der Entwicklung nach den Methoden der objektorientierten Softwareentwicklung vor (analog zum Programmentwurf des TicTacToe-Spiels in UE 26).

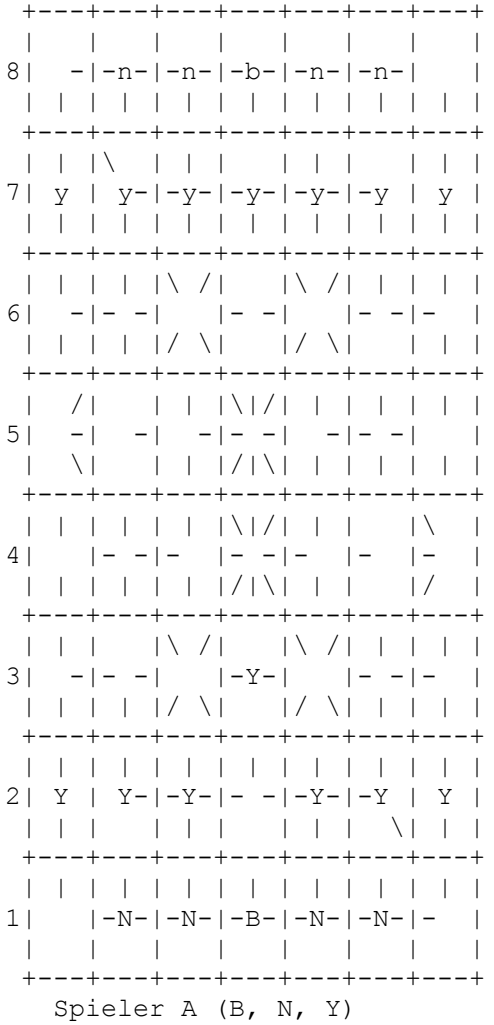
Skizze eines möglichen Programmablaufs:





```
   Spieler A (B, N, Y)

Spieler A ist am Zug!
Von Reihe (1..8): 2
Von Spalte (a..g):d
Nach Reihe (1..8): 3
Nach Spalte (a..g):d
```



```
   Spieler A (B, N, Y)

Spieler B ist am Zug!
```

```
Von Reihe (1..8):

...
```

Aufgabe 10:

Bei dieser Aufgabe sollen Sie ein Programm entwickeln, durch das ein menschlicher Spieler am Computer das Knobelspiel *Lampen* spielen kann.

Regeln

Das Spiel besteht aus einem quadratischen Spielfeld der Größe *n* ($2 < n < 10$), das aus einzelnen Zellen besteht, die Lampen repräsentieren. Lampen können sich im Zustand *an* oder *aus* befinden. Anfangs sind alle Lampen im Zustand *aus*. In jedem Spielzug wählt der Spieler eine Zelle aus, deren Zustand umgeschaltet werden soll (von *aus* in *an* oder umgekehrt). Gleichzeitig werden aber auch die Nachbarlampen oberhalb, unterhalb, links und rechts von der entsprechenden Lampe umgeschaltet (insofern sie existieren). Ziel (und Ende) des Knobelspiels ist es, den Zustand zu erreichen, dass alle Lampen angeschaltet sind.

Aufgabe
Schreiben Sie ein Java-Programm, mit dessen Hilfe ein menschlicher Spieler das Spiel *Lampen* spielen kann. Gehen Sie bei der Entwicklung nach den Methoden der objektorientierten Softwareentwicklung vor (analog zum Programmentwurf des TicTacToe-Spiels in UE 26). Finden Sie geeignete Klassen und Methoden.

Hinweise:

Der generelle Spielablauf lässt sich folgendermaßen skizzieren:

- Abfrage der Spielfeldgröße
- Initialisierung des Spielfeldes
- Ausgabe des Spielfeldes
- Solange das Spiel nicht beendet ist, tue folgendes
    - Korrekten Spielzug einlesen
    - Spielzug ausführen
    - Ausgabe des Spielfeldes

Beispiel für einen Programmablauf (Benutzereingaben stehen in Klammern (<>), Lampen im Zustand *aus* werden durch ein ‚.', Lampen im Zustand *an* durch ein ‚+' gekennzeichnet):

```
Feldgroesse (2 < n < 10): <5>

 01234
0.....
1.....
2.....
3.....
4.....
```

```
Reihe der Lampe, die umgeschaltet werden soll (0 <= r < 5): <3>
Spalte der Lampe, die umgeschaltet werden soll (0 <= r < 5): <2>
 01234
0.....
1.....
2..+..
3.+++.
4..+..

Reihe der Lampe, die umgeschaltet werden soll (0 <= r < 5): <0>
Spalte der Lampe, die umgeschaltet werden soll (0 <= r < 5): <0>

 01234
0++...
1+....
2..+..
3.+++.
4..+..

Reihe der Lampe, die umgeschaltet werden soll (0 <= r < 5): <3>
Spalte der Lampe, die umgeschaltet werden soll (0 <= r < 5): <1>

 01234
0++...
1+....
2.++..
3+..+.
4.++..

...
```

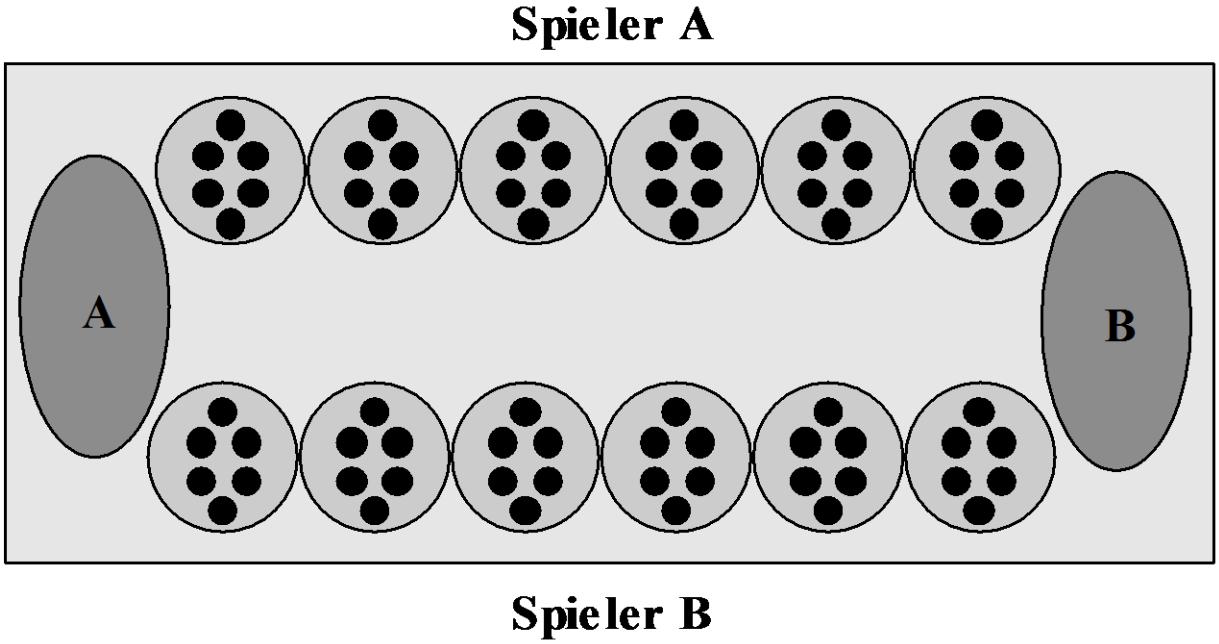
Aufgabe 11:

Kalah ist ein Spiel für zwei Personen. Es besteht aus einem Brett mit zwei Reihen von je sechs Löchern, in denen anfangs je sechs Kugeln liegen, und zwei weiteren Löchern, die Kalah heißen und anfangs leer sind (siehe Abbildung). Spieler B gehören die unteren Löcher und das rechte Kalah (B), Spieler A die oberen Löcher und das linke Kalah (A).

Ein Zug läuft wie folgt ab:

- Der Spieler entleert eines seiner nicht leeren Löcher (nicht sein Kalah) und verteilt auf die entgegen dem Uhrzeigersinn folgenden Löcher je eine Kugel, wobei er das Kalah des Gegners auslässt.
- Landet die letzte Kugel im eigenen Kalah, so darf und muss der Spieler noch einmal ziehen.
- Landet die letzte Kugel in einem eigenen leeren Loch und ist das gegenüberliegende Loch des Gegners nicht leer, so kommt die Kugel mit den gegenüberliegenden Kugeln ins eigene Kalah.

Beide Spieler ziehen abwechselnd, bis alle Löcher eines Spielers leer sind. Dann entleert der andere Spieler seine Löcher in sein Kalah. Gewonnen hat der Spieler, dessen Kalah am Ende mehr Kugeln enthält.



Aufgabe: Implementieren Sie das *Kalah*-Spiel, so dass zwei Menschen gegeneinander *Kalah* spielen können! Gehen Sie bei der Entwicklung nach den Methoden der objektorientierten Softwareentwicklung vor (analog zum Programmentwurf des TicTacToe-Spiels in UE 26).

Aufgabe 12:

Reversi ist ein Spiel für zwei Personen.

Spielbrett

Reversi wird auf einem Spielbrett gespielt, das sich aus 8*8 einzelnen Feldern zusammensetzt. Es hat also dieselben Ausmaße wie ein Schachbrett, nur das die Felder nicht abwechselnd schwarz und weiß sind, sondern eine einheitliche Farbe haben. Es ist zweckmäßig, das Brett wie beim Schach mit Koordinaten zu versehen, um Spielzüge eindeutig angeben zu können.

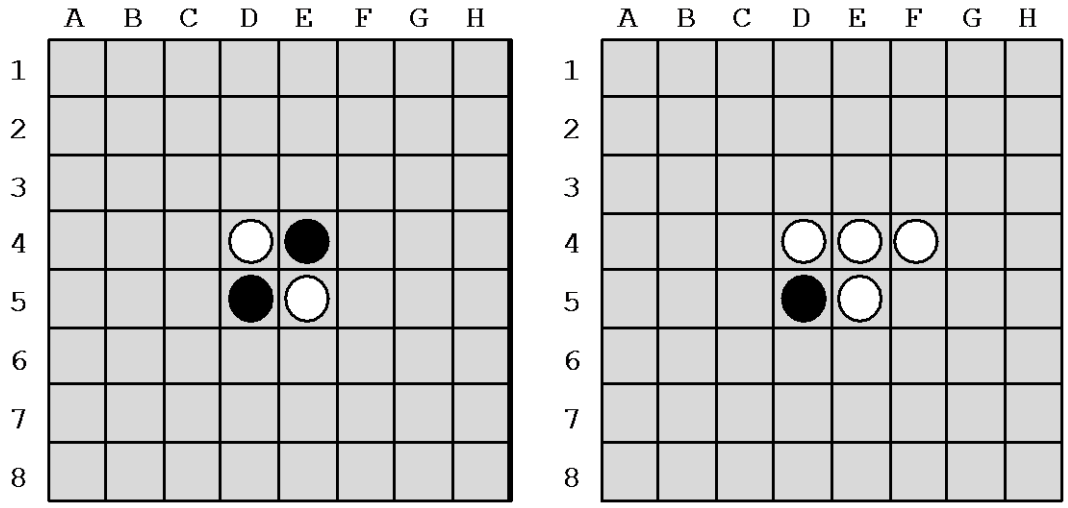
Spielfiguren

Als Spielfiguren werden runde Plättchen (Steine) verwendet. Diese haben je eine schwarze und eine weiße Seite. Es gibt insgesamt 64 Plättchen, also genausoviele, wie das Spielbrett Felder hat. Die Spielplättchen werden im Laufe des Spiels umgedreht (daher der Name Reversi), so daß schwarze zu weißen Steinen werden können und umgekehrt. Im folgenden ist immer von *schwarzen* und *weißen Steinen* die Rede, was sich jeweils auf die Oberseite der Steine - also die sichtbare Farbe - bezieht.

Spielverlauf und Ziel des Spiels

Der Spieler, der das Spiel beginnt, wird im folgenden *Spieler A* genannt. Der andere ist dementsprechend *Spieler B*. Spieler A bekommt die Farbe „Weiß“, Spieler B die Farbe „Schwarz“ zugeteilt. Zu Anfang des Spiels befinden sich vier Steine auf dem Spielbrett, zwei mit ihrer weißen (D4 und E5), zwei mit ihrer schwarzen Seite (E4 und

D5) nach oben (siehe Abbildung links). Die restlichen 60 Steine liegen in einem Sammelpool neben dem Spielbrett.



Spieler A führt dann seinen ersten Spielzug aus: Er nimmt einen Stein aus dem Sammelpool und legt ihn auf ein leeres Feld (selbstverständlich mit seiner Farbe „Weiß“ nach oben). Das leere Feld muß dabei an ein belegtes Feld horizontal, vertikal oder diagonal angrenzen. Außerdem muß der Spielzug zum „Schlagen“ von mindestens einem gegnerischen Stein führen. „Schlagen“ bedeutet für Spieler A: Drehe alle schwarzen Steine um, die sich horizontal, vertikal oder diagonal zwischen bereits gesetzten weißen Steinen und dem neu gesetzten (ebenfalls weißen) Stein befinden. In der Ausgangsituation kann sich Spieler A also eines der Felder E3, F4, D6 und C5 aussuchen. Setzt er den Stein bspw. auf Feld F4, muß er den schwarzen Stein auf Feld E4 umdrehen (siehe Abbildung rechts).

Damit ist der erste Spielzug von Spieler A beendet und Spieler B kommt zum Zug. Dieser führt nun eine identische Aktion durch, nur daß er jetzt natürlich weiße Steine schlagen muß.

Die beiden Spieler führen abwechselnd ihre Spielzüge durch. Ist es einem Spieler nicht möglich, ein leeres Feld mit einem Stein zu besetzen, so muß er passen und der andere Spieler ist wieder am Zug.

Das Spiel ist beendet, wenn kein Spieler mehr einen Stein setzen kann. Dies ist in der Regel der Fall, wenn alle Felder besetzt sind, kann aber auch schon vorher passieren. Die schwarzen und weißen Steine auf dem Spielbrett werden gezählt. Sieger des Spiels ist der Spieler, der die größere Anzahl an Steinen auf dem Spielbrett hat.

Achtung

- Ein Stein, der einmal auf dem Spielbrett liegt, wird nie mehr vom Brett genommen oder verschoben. Er wird höchstens umgedreht.
- Jeder Spielzug muß vollständig ausgeführt werden, d.h. alle Steine, die aufgrund eines neu gesetzten Steines geschlagen werden können, müssen auch umgedreht werden. Die Spieler dürfen sich nicht aussuchen, welche Steine sie umdrehen und welche nicht.

- Solange ein Spieler gegnerische Steine schlagen kann, muß er ziehen. Er darf nicht freiwillig passen.
- Der entscheidende Stein beim Umdrehen ist der neu gesetzte Stein. Zum Beispiel werden weiße Steine nicht umgedreht, die zwischen zwei schwarzen Steinen liegen, die beide schon vorher auf dem Spielbrett lagen. Es findet also keine transitive Fortsetzung beim Umdrehen von Steinen statt. Wird bspw. bei der Ausgangsposition in der Abbildung unten links ein schwarzer Stein auf Feld F6 gesetzt, dann werden die beiden weißen Steine auf den Feldern D4 und E5 umgedreht, und es ergibt sich die Situation in der Abbildung unten rechts. Obwohl durch die Umdreh-Aktion nun auch der weiße Stein auf Feld D6 zwischen zwei schwarzen Steinen (E5 und C7) liegt, wird dieser nicht umgedreht, weil sich die beiden schwarzen Steine schon vorher auf dem Spielbrett befanden.

Aufgabe: Implementieren Sie das *Reversi*-Spiel, so dass zwei Menschen gegeneinander *Reversi* spielen können! Gehen Sie bei der Entwicklung nach den Methoden der objektorientierten Softwareentwicklung vor (analog zum Programmentwurf des TicTacToe-Spiels in UE 26).

Aufgabe 13:

Metamorphose ist ein Spiel für zwei Personen (Spieler A und Spieler B).

Spielbrett

Metamorphose wird auf einem Spielbrett gespielt, das sich aus 7*7 einzelnen Feldern zusammensetzt. Bestimmte Felder sind durch Mauern besetzt (siehe Abbildung). Es ist zweckmäßig, das Brett wie beim Schach mit Koordinaten zu versehen, um Spielzüge eindeutig angeben zu können.

Spielfiguren

Als Spielfiguren werden Quadrate und Kreise verwendet. Es gibt weiße Spielfiguren, die Spieler A gehören, und schwarze Spielfiguren, die Spieler B gehören. Anfangs besitzt jeder Spieler vier Quadrate und drei Kreise.

Spielzug

In jedem Spielzug verschiebt ein Spieler eine seiner Spielfiguren um ein oder mehrere Felder auf dem Spielbrett. Dabei gilt: Quadrate können horizontal und vertikal verschoben werden. Kreise können diagonal verschoben werden. Nachdem eine Figur verschoben wurde, wechselt sie ihren Typ (aus einem Kreis wird ein Quadrat und umgekehrt).

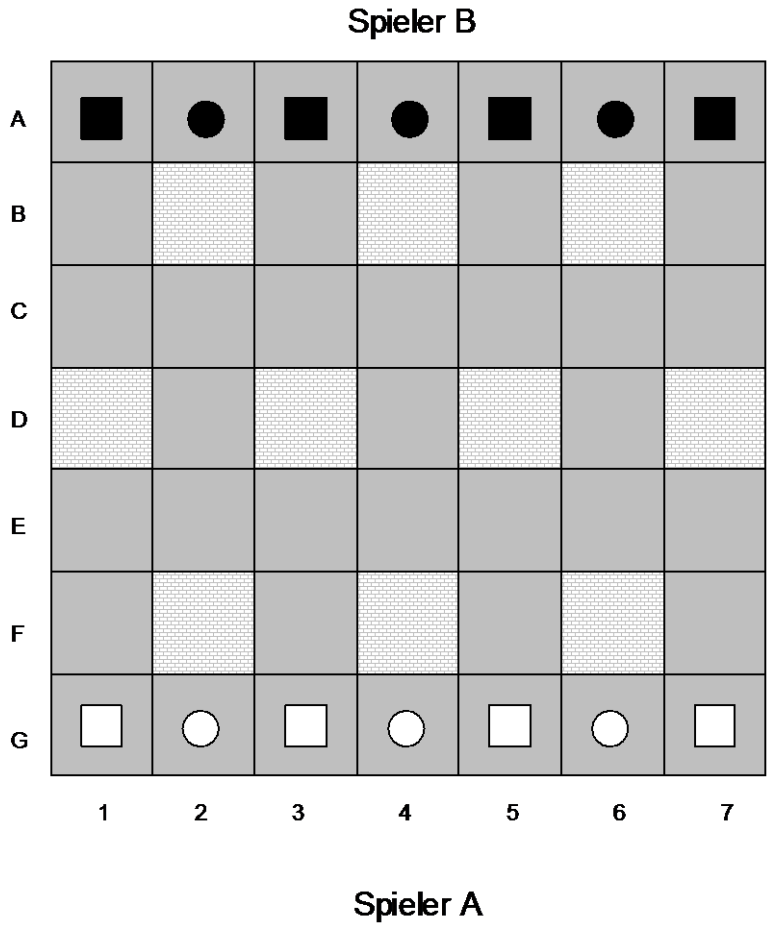
Spielregeln

- Die Anfangsaufstellung ist in der Abbildung dargestellt.
- Spieler A (Weiß) beginnt.
- Es besteht Zugzwang. Kann ein Spieler nicht mehr ziehen kann, hat er verloren!
- Auf einem Feld darf maximal eine Figur stehen.

- Auf die durch Mauern besetzten Felder dürfen keine Figuren gezogen werden.
- Bei einem Spielzug dürfen keine Mauern und andere Figuren (der eigenen oder fremden Farbe) übersprungen werden.
- Endet ein Spielzug auf einem Feld, auf dem eine fremde Figur steht, so gilt diese als geschlagen und wird vom Spielbrett entfernt.
- Eigene Figuren dürfen nicht geschlagen werden.

Ziel des Spiels

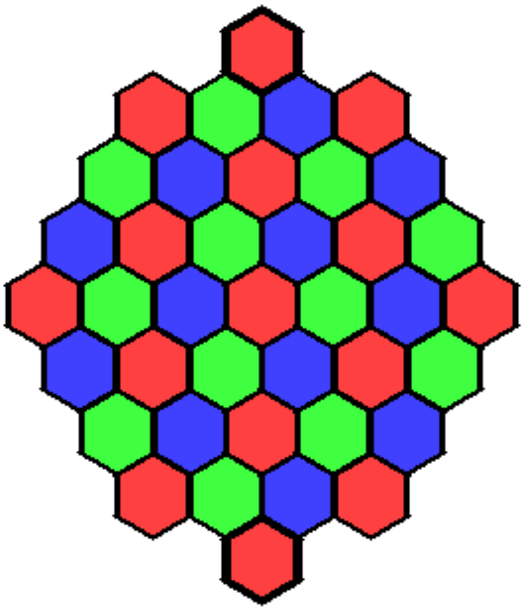
Am Anfang des Spiels wählt jeder Spieler (geheim) eine seiner Spielfiguren aus und merkt sie sich als besondere Spielfigur, den König. Wer als erster den König des Gegners schlägt, hat gewonnen. Das Spiel endet unentschieden, wenn beide Spieler nur noch ihren König besitzen.



Aufgabe: Implementieren Sie das *Metamorphose*-Spiel, so dass zwei Menschen gegeneinander *Metamorphose* spielen können! Gehen Sie bei der Entwicklung nach den Methoden der objektorientierten Softwareentwicklung vor (analog zum Programmentwurf des TicTacToe-Spiels in UE 26).

Aufgabe 14:

PAROB ist eine Schachvariante, die auf einem Spielbrett mit 39 Feldern gespielt wird, deren Anordnung in der folgenden Abbildung gezeigt wird. Bis auf die Ausnahmen, die im folgenden aufgelistet werden, sind die Spielregeln dieselben wie beim Schach.



Ziel des Spiels

Die roten Felder mit den dicken Linien ganz unten und oben sind die „Burgen“ der beiden Spieler. Ziel des Spiel ist es, eine eigene Spielfigur in der gegnerischen Burg zu platzieren.

Spieler

PAROB ist ein Spiel für zwei Spieler. Spieler Weiß spielt mit weißen Spielfiguren von unten nach oben. Spieler Schwarz spielt mit schwarzen Spielfiguren von oben nach unten. Es wird immer abwechselnd gezogen, wobei Spieler Weiß beginnt.

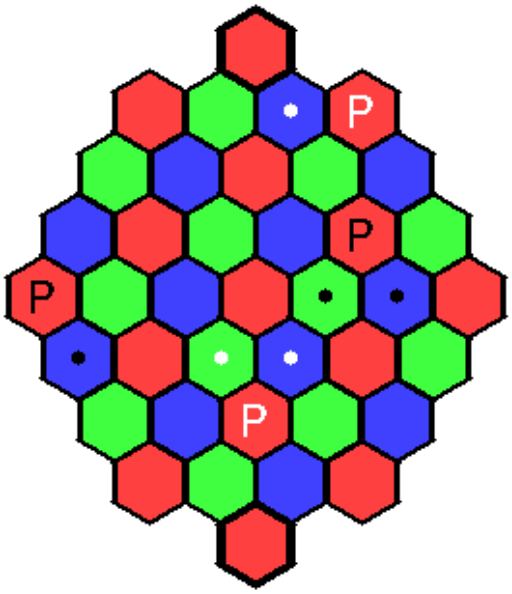
Spielfiguren

Es gibt drei Typen von Spielfiguren: Bauern, Türme und Läufer. Eigene Spielfiguren können nicht geschlagen werden. Auf jedem Feld des Spielbretts kann höchstens eine Spielfigur stehen.

Bauern

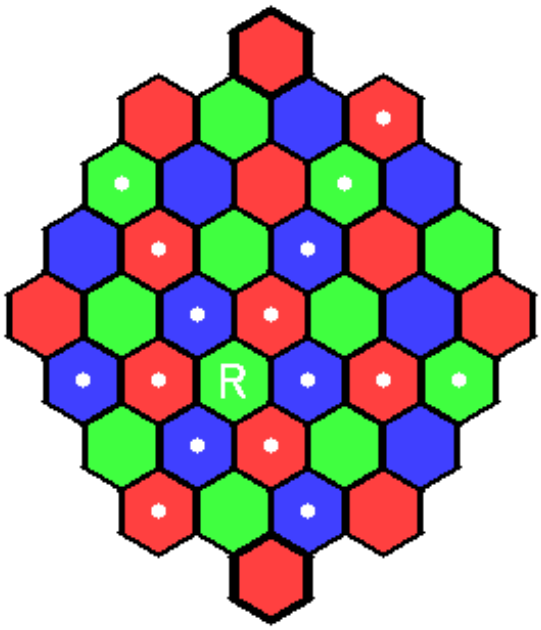
Ein Bauer (P) kann in einem Spielzug verschoben werden auf das Feld oberhalb von ihm links und oberhalb von ihm rechts (mit oberhalb ist damit gemeint: weg von der eigenen Burg). Falls ein Bauer auf einem der beiden roten Felder in der vorletzten Reihe steht, kann er auch auf das Feld links bzw. rechts daneben verschoben werden. Steht auf dem Feld, auf das ein Bauer verschoben wird, eine gegnerische

Figur, so gilt diese als geschlagen und wird vom Spielbrett entfernt. Anders als beim Schachspiel schlagen Bauern genauso wie sie bewegt werden. Sie dürfen auch nicht wie beim Schach anfangs zwei Felder vorwärts bewegt werden. Die folgende Abbildung enthält Beispiele für gültige Bauernzüge.



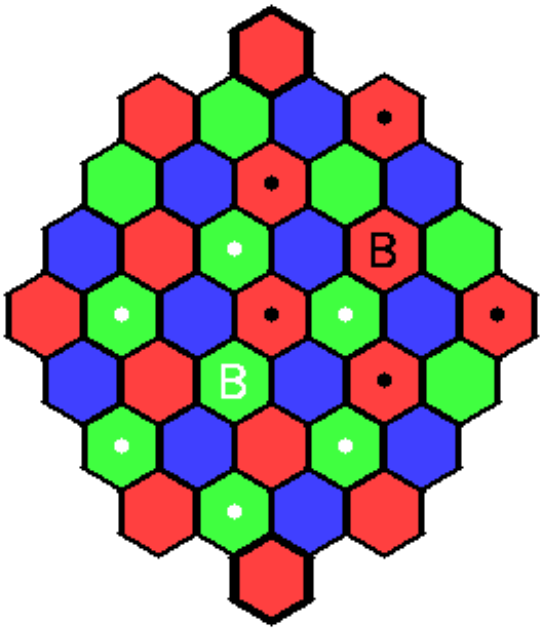
Türme

Ein Turm (R) kann in einem Spielzug eine beliebige Anzahl von Feldern in einer geraden Linie in alle sechs Richtungen (links, rechts, oben-links, oben-rechts, unten-links, unten-rechts) verschoben werden. Er darf dabei weder eigene noch fremde Spielfiguren überspringen. Steht auf dem Feld, auf das ein Turm verschoben wird, eine gegnerische Figur, so gilt diese als geschlagen und wird vom Spielbrett entfernt. Die folgende Abbildung enthält Beispiele für gültige Turmzüge.



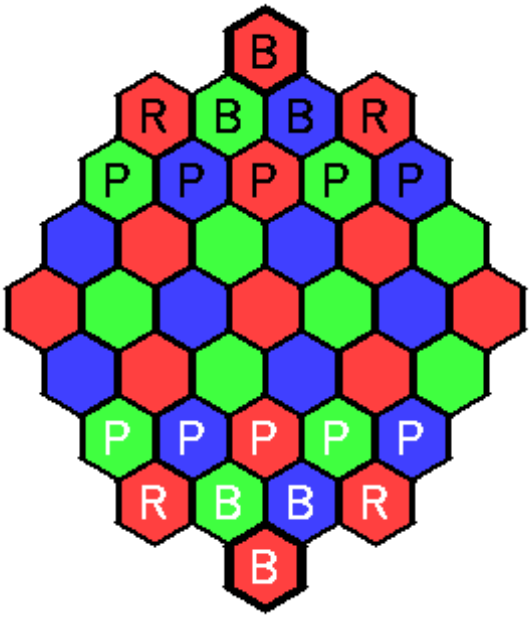
Läufer

Ein Läufer (B) kann in einem Spielzug verschoben werden auf ein beliebiges der sechs Felder, die zwei Felder entfernt sind und dieselbe Farbe haben (Läufer bleiben immer auf Feldern derselben Farbe). Läufer können dabei auch eigene und fremde



Spielfiguren überspringen (sie sind also quasi eine Mischung von Läufern und Springern beim Schach). Die folgende Abbildung enthält Beispiele für gültige Läuferzüge.

Spielbeginn



Die folgende Abbildung zeigt die initiale Aufstellung der Spielfiguren.

Zusatzregeln

- Sobald ein Spieler keine Spielfiguren mehr hat, die die gegnerische Burg besetzen können, hat er verloren.
- Sobald ein Spieler keine legalen Spielzüge mehr ausführen kann, hat er verloren.

Aufgabe: Implementieren Sie das *Parob*-Spiel, so dass zwei Menschen gegeneinander *Parob* spielen können! Gehen Sie bei der Entwicklung nach den Methoden der objektorientierten Softwareentwicklung vor (analog zum Programmentwurf des TicTacToe-Spiels in UE 26).

Aufgabe 15:

Rushhour ist ein Spiel für einen einzelnen Spieler. Auf einem rechteckigen Spielbrett, das aus einzelnen Kacheln besteht, wird initial eine Menge an Autos platziert. Jedes Auto überdeckt mindestens zwei Kacheln. Ein Auto mit mehr als zwei Kacheln darf dabei keine Ecken aufweisen. Liegen die Kacheln, die ein Auto überdeckt, nebeneinander, ist es ein horizontal verschiebbares Auto. Liegen die Kacheln übereinander, ist es ein vertikal verschiebbares Auto. Ein Auto wird als Hauptauto gekennzeichnet. Ziel des Spiels ist es nun, die Autos so zu verschieben, bis irgendwann das Hauptauto (im Bild bspw. das rote Auto) den rechten Rand berührt.

Horizontale Autos dürfen dabei nur nach links und rechts, vertikale Autos nur nach oben und unten verschoben werden. Ein Auto darf nicht über den Rand hinweg oder wenn ein anderes Auto im Wege steht, verschoben werden.

Entwickeln Sie ein Java-Programm, mit dem ein Benutzer Rushhour spielen kann. Die Ausgangsstellung der Autos sei dabei in einer Textdatei festgelegt, die mittels einer der readFile-Methoden der Klasse IO gelesen werden kann. Die Autoteile werden dabei durch jeweils gleiche Zeichen repräsentiert; das Hauptauto durch das Zeichen ‚*'. Die Stellung der Autos in der oberen Abbildung entspräche bspw. folgendem Dateiinhalt:

```
122 3
145 36
145**6
7778 6
  98aa
bb9cc
```

Der Benutzer soll nun solange Autos verschieben können (durch Angabe von Auto und Richtung) bis das Hauptauto den rechten Rand erreicht. Im Folgenden wird ein beispielhafter Programmablauf skizziert (Benutzereingaben in <>):

```
122 3
145 36
145**6
7778 6
  98aa
bb9cc

Wahl eines Autos (Buchstabe angeben): <c>
Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <>r
122 3
145 36
145**6
7778 6
  98aa
bb9 cc

Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <q>
Wahl eines Autos (Buchstabe angeben): 8
Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <d>
122 3
145 36
145**6
777  6
  98aa
bb98cc

Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <q>
Wahl eines Autos (Buchstabe angeben): 7
Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <r>
122 3
145 36
145**6
 777 6
  98aa
bb98cc

Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <r>
122 3
145 36
145**6
  7776
  98aa
bb98cc

...
```

Gehen Sie bei der Entwicklung nach den Methoden der objektorientierten Softwareentwicklung vor (analog zum Programmentwurf des TicTacToe-Spiels in UE

26). Definieren Sie bspw. die Klassen `Spiel`, `Spielbrett` und `Auto`, jeweils mit geeigneten Methoden.