Allgemeine Informatik 1
Sommersemester 2023

Programmierprojekt, Version: 5. Juni 2023

TECHNISCHE UNIVERSITÄT DARMSTADT

Bearbeitungszeit: 07.06.2023 bis 05.07.2023

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/mod/assign/view.php?id=55356`
- 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 Ihres*r Partners*in!) ein.
- Um das ProPro abzugeben, müssen Sie in Moodle in einer ProPro-Gruppe angemeldet sein. Die Abgabe in Moodle erfolgt immer für die gesamte Gruppe. Bei Zweiergruppen ist es ausreichend, wenn die Lösung nur von einer Person hochgeladen wird.
- Abgabe des Projekts (spätestens bis zum **05.07.2023** um **22:00 Uhr**)
  - Verpacken Sie Ihr gesamtes Projektverzeichnis in einer Standard-ZIP-Datei. Der Name der Datei muss aus den Initialen der Gruppenmitglieder und der verwendeten Entwicklungsumgebung bestehen, z. B. **MK_bluej.zip** bzw. **MK_eclipse.zip**. Bei Zweiergruppen entsprechend beide Namen, z. B. **MK_CK_bluej.zip**.
  - Laden Sie die ZIP-Datei auf unserer Moodle-Plattform hoch. **Keine** ZipX-, 7z- oder RAR-Archive! Eine genauere Anleitung zum Erstellen der Abgabe finden Sie auf Moodle im Bereich des Programmierprojekts.
  - 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 25 Punkte zu erreichen. Mängel bei Formatierung oder Kommentierung führen zu Abzügen (siehe 6). Nicht kompilierbare Abgaben erhalten **0 Punkten**. Das Projekt sollte idealerweise mit **BlueJ** oder **Eclipse** bearbeitet werden. Es ist erlaubt, eine andere IDE zu verwenden, solange das Projekt unter **BlueJ** oder **Eclipse** ausgeführt werden kann und die README.txt-Datei entsprechend enthalten ist.
- **Achtung:** Der Fachbereich Informatik legt großen Wert auf die Einhaltung wissenschaftlicher Ethik, einschließlich des strikten Verbots von Plagiaten. Mit der Abgabe Ihrer Lösung bestätigen Sie, dass Sie der*die alleinige Autor*in des gesamten Materials sind. Das bedeutet, dass „Abschreiben“ (auch verfremdet), „Abschreiben lassen“ und das “Vergleichen von Lösungen“, sei es durch Weitergabe von Programmcode oder mündlich, verboten sind. Achten Sie darauf, Ihr Notebook, Ihren Rechner oder Ihren USB-Stick nicht unbeaufsichtigt zu lassen. Weitere Informationen zu diesem Thema finden Sie 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 **1**, **2** und **3** selbst schreiben sollen.

2) Sie können das Projekt in der gewohnten Entwicklungsumgebung **BlueJ** bearbeiten. Gerne können Sie auch die Gelegenheit nutzen, um **Eclipse** (eine leistungsfähigere IDE) kennenzulernen und das Projekt hierin bearbeiten. Weitere Informationen zu den Unterschieden zwischen **BlueJ** und **Eclipse** sowie den Vorteilen von **Eclipse** finden Sie im Moodle-Forum.

3) Das Projekt erfordert mindestens **Java Version 11**. Für **Eclipse** oder **BlueJ** ist keine vorinstallierte Java Version mehr erforderlich. In der Regel können Sie einfach die neuesten Java Versionen von **Eclipse** oder **BlueJ** verwenden. Falls es dabei Probleme gibt, können Sie auch eine eigenständige Java Version installieren und verwenden. Überprüfen Sie Ihre installierte Java Version auf folgende Weise:

   - **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/de/java/technologies/downloads/`

   Hinweis für Mac-Nutzer*innen mit Apple Silicon (bspw. M1): Obwohl es inzwischen angepasste Java Versionen für Apple Silicon gibt, muss für dieses Projekt in **Eclipse** dennoch eine Intel-Variante genutzt werden. Das Einbinden gestaltet sich etwas komplizierter, daher gibt es auf Moodle ein Tutorial im Abschnitt **ProPro Einrichtung**.

   Hinweis für Linux-Nutzer*innen: Wir haben die Erfahrung gemacht, dass es unter **Linux** Probleme mit der Ausführung dieses Projekts geben kann. Stellen Sie sicher, dass Sie die neusten Versionen von **Eclipse** oder **BlueJ** verwenden. Ebenfalls sollten Sie keine vorinstallierte Java Version, sondern die aktuelle Java Version Ihrer IDE nutzen. Wenn das Programmierprojekt so nicht laufen sollte, haben wir für Sie eine separate **Linux-Anleitung** erstellt, mit der Sie eine bestimmte Java Version installieren und mit dieser arbeiten können. Die **Linux-Anleitung** finden Sie im Moodle-Forum im Bereich des Programmierprojekts.

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. Sie müssen nun noch die benötigten Bibliotheken hinzufügen. Klicken Sie hierfür auf:

     Tools → *Preferences…* → *Libraries* → *Add File*

     Im entpackten Projektordner liegen im Unterordner **libs** die drei benötigten Dateien **eea.jar**, **slick.jar** sowie **lwjgl.jar**. Fügen Sie diese hinzu! Danach muss BlueJ neu gestartet werden. Haben Sie dies getan, können Sie nun mit *Project* → *Open Project…* den ausgepackten Projektordner wählen und das Projekt so hinzufügen.

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

Vorgaben

Nachdem Sie die wichtigen Hinweise beachtet haben, sollten Sie nun das Programmierprojekt öffnen können. Es enthält alle benötigten Klassen in Unterpaketen des Pakets `tud.ai1.shisen`. Während des Programmierprojekts werden Sie nur in den Unterpaketen **model** und **util** 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.

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! Einige Methoden sind noch nicht implementiert bzw. enthalten nur eine „Dummy-Implementierung“. Es ist im Folgenden Ihre Aufgabe diese Methoden zu vervollständigen (und auf Kommentierung sowie Formatierung zu achten).

Alle anderen Methoden und Klassen der Vorgabe müssen unangetastet bleiben, auch sind keine zusätzlichen Datenfelder jedweder Sichtbarkeit erlaubt!

Einleitung - Das Spiel *Shisen*

Shisen ist eine stark abgeänderte Variante des chinesischen Brettspiel-Klassikers Mahjongg. Das Spiel umfasst 36 unterschiedliche Felder, die jeweils vier Mal auf dem Spielfeld platziert werden. Das Spielfeld setzt sich somit aus insgesamt 144 Feldern zusammen, angeordnet in einer 8x18-Matrix. Ziel des Spiels ist es, alle Felder vom Spielfeld zu entfernen, wobei immer zwei identische Felder gleichzeitig entnommen werden können. Zwei Felder können genau dann entnommen werden, wenn sie mit maximal drei Linien verbunden werden können. Die Linien müssen entweder horizontal oder vertikal (nicht diagonal) zum Spielfeld sein und dürfen keine anderen Felder kreuzen. Dies bedeutet insbesondere, dass benachbarte Felder direkt entnommen werden können, unabhängig davon, wo sie sich auf dem Spielfeld befinden. Linien dürfen den Spielfeldrand um ein Feld überschreiten.

Hinweis: Wundern Sie sich nicht, wenn ein Spiel nicht komplett gelöst werden kann. Dadurch, dass zwei Paare jedes Feldes vorhanden sind, ist es bei ”falschem” Lösen eines Paares unter Umständen möglich, dass das Spiel nicht komplett lösbar ist.

Weitere Informationen zum Spiel sowie eine Variante können Sie hier finden[2]:

Erläuterungen: `https://dkmgames.com/shisen/`
Beispiel: `https://dkmgames.com/shisen/shisenplay.htm`

Beachten Sie, dass die genauen Spezifikationen der Aufgabenstellung entnommen werden muss.

Im Hauptmenü (Abbildung 1a) können Sie ein neues Spiel starten oder sich die *Highscores* ansehen. Möchten Sie ein neues Spiel starten, klicken Sie einfach auf die Schaltfläche *Start Game*. Auf dem Spielfeld selbst (Abbildung 1b) können Sie mit der linken Maustaste die Karten umdrehen oder mit den Schaltflächen *Hint*, *Solve Pair* oder *Find Partner* einen Cheat aktivieren, der Hinweise gibt oder beispielsweise ein passendes Paar aufdeckt. Aktivieren sie ein Feld und klicken anschließend auf einen passenden Partner, so verschwinden beide Felder (Abbildung 1c). Zusätzlich sehen Sie oben rechts die bereits verstrichene Zeit sowie die von Ihnen erzielte Punktezahl (Score). In der Highscore-Übersicht sind die besten zehn Einträge zu finden (Abbildung 1d).

Hinweis: Um das Spiel zu starten kompilieren sie alle Quellcode-Dateien in den Unterpaketen und starten sie die **`main`**-Methode in der Klasse **`Launch`** im Unterpaket **`view`**.

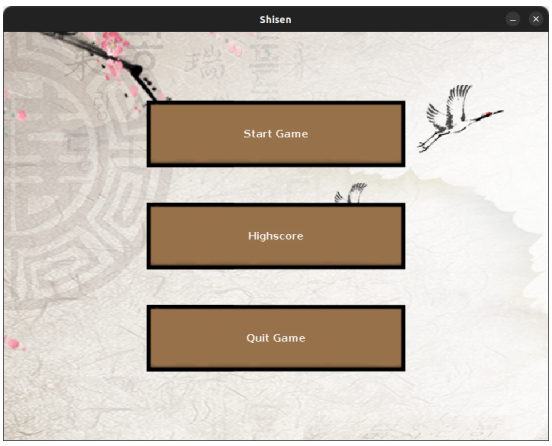
1 Aufzählungstyp (0.25 Punkte)

Wie bereits erwähnt ist das gesamte Projekt noch nicht kompilierbar. Daher werden auch noch einige Fehler im Quellcode angezeigt. Das liegt unter anderem daran, dass der Aufzählungstyp **`TokenState`** noch nicht vorhanden ist. In dieser Aufgabe sollen Sie diesen Typ implementieren.
Der Aufzählungstyp **`TokenState`** steht für den Zustand eines Feldes und wird für die Markierung dieses genutzt.
Ein Feld kann folgende vier Zustände annehmen: *Ausgangszustand*, *Angeklickt*, *Falsch* und *Gelöst*.

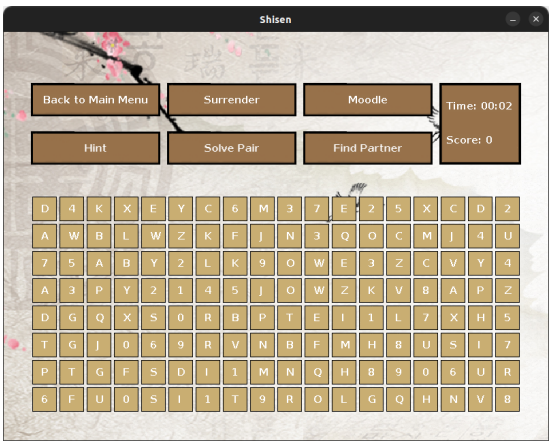
a) Erstellen Sie im Paket **`model`** einen Aufzählungstyp (= **`enum`**) mit dem Namen **`TokenState`**. Dieser soll folgende Werte besitzen:

`DEFAULT`, **`CLICKED`**, **`WRONG`**, und **`SOLVED`**.

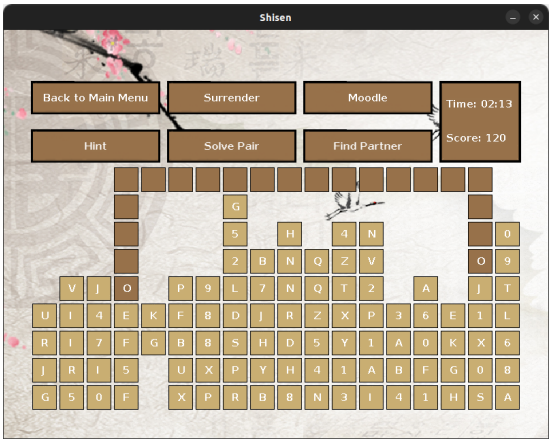
[2]Bitte beachten Sie jedoch, dass die verlinkten Seiten eine spezielle Beschreibung des Spiels für eine genaue Implementierung beschreiben. Unser Projekt folgt nicht zwangsweise dieser genauen Beschreibung!



(a) Hauptmenü



(b) Spielfeld nach Starten des Spiels



(c) Spielfeld während des Spielens

(d) Highscores

Abbildung 1: Übersicht der einzelnen Spielzustände.

2 Token (0.25 + 0.25 + 0.25 + 0.5 + 0.5 + 0.25 = 2 Punkte)

In dieser Aufgaben sollen Sie nun die Klasse **`Token`** erstellen, welche für ein einzelnes Feld auf dem gesamten Spielfeld steht.

a) Erstellen Sie im Paket **`model`** die Klasse **`Token`**, welche das Interface[3] **`IToken`** implementiert und fügen Sie dieser folgende fünf private Instanzvariablen hinzu:

- **`counter`**: Diese Variable soll vom Typ **`int`** und statisch sein. Mit ihr werden eindeutige Ids generiert, weswegen sie direkt bei der Deklaration auch mit **`0`** initialisiert werden muss.
- **`id`**: Eine **`int`** Variable, die außerdem unveränderbar sein soll. Durch diese Variable ist das einzelne Feld auf dem gesamten Spielfeld eindeutig zu identifizieren.
- **`state`**: Ein Datenfeld des Typs **`TokenState`**. Es beschreibt den aktuellen Zustand des Feldes.
- **`value`**: Diese Variable soll vom Typ **`int`** und unveränderbar sein. Sie beschreibt den Wert eines Feldes.
- **`pos`**: Eine **`Vector2f`** Variable, die im Spiel den Ort des einzelnen Feldes auf dem Spielfeld angibt.

[3] An dieser Stelle greift das Programmier-Projekt etwas aus der Vorlesung zum Thema Vererbung vor. Ein Interface sorgt dafür, dass Klassen vordefinierte Methoden bereitstellen. Um eine Klasse von einem Interface erben zu lassen, muss im Klassenkopf nach dem Namen der Klasse das Schlüsselwort **`implements`** und anschließend der Name des Interface folgen. Jede nicht im Interface implementierte aber deklarierte Methode muss in der erbenden Klasse implementiert werden.

Hinweis: Der statische Typ der Variable `pos` soll `org.newdawn.slick.geom.Vector2f` sein. (Die Variable `pos` soll allerdings **nicht** statisch sein!) Achten Sie auf nötige Import-Anweisungen!

b) Erstellen Sie einen Konstruktor der Klasse `Token`. Dieser soll nur drei Parameter besitzen, welche vom Typ `int`, `TokenState` und `Vector2f` sind. Mit diesen drei Parametern initialisieren Sie die Instanzvariablen `value`, `state` und `pos`. Beachten Sie, dass Sie in diesem Konstruktor auch den Wert des Datenfeldes `id` auf den Wert von `counter` setzen müssen und `counter` anschließend um `1` erhöhen.

c) Um bei der Erstellung eines Objekts dieser Klasse nicht immer alle Werte angeben zu müssen, soll zusätzlich zu dem in 2b erstellten Konstruktor noch ein weiterer implementiert werden, welcher nur einen `int` Parameter erwartet. Dieser Parameter gibt den Wert für die Instanzvariable `value` an. Weiterhin muss das Datenfeld `state` mit `TokenState.DEFAULT` sowie `pos` mit einem neuen Objekt des Typs `Vector2f` mit den Parametern `0, 0` initialisiert werden. Die Datenfelder `id` und `counter` müssen genau wie in 2b initialisiert werden.

Hinweis: Aus einem Konstruktor heraus können auch andere Konstruktoren der gleichen Klasse aufgerufen werden.

d) Fügen Sie der Klasse die vier getter-Methoden `getValue()`, `getTokenState()`, `getID()` und `getPos()` hinzu. Diese sollen die entsprechende Instanzvariable zurückgeben.

Hinweis: Ist eine Methode, welche in der Klasse implementiert wurde, schon im Interface deklariert, so muss diese zum Überschreiben in der Klasse mit der Annotation `@Override` gekennzeichnet werden.

e) Schreiben Sie die getter-Methode `getDisplayValue()`, welche das *DisplayValue* des `TokenDisplayValueProvider`s zurückgibt.

Hinweis: Der `TokenDisplayValueProvider` ist als ein *Singleton*-Designpattern implementiert. Das bedeutet, dass keine Objekte der Klasse erstellt werden können, sondern bei jeder Verwendung mit der Methode `TokenDisplayValueProvider.getInstance()` die Instanz des Objekts geholt werden muss, um Operationen durchführen zu können.

Hinweis: Auch diese Methode ist schon im Interface `IToken` deklariert. Dementsprechend muss auch hier eine Annotation eingesetzt werden.

Hinweis: Achten Sie auf nötige Import-Anweisungen!

f) Schreiben Sie die beiden setter-Methoden `setTokenState(TokenState newState)` und `setPos(Vector2f newPos)`. Die Setter setzen die jeweils passende Instanzvariable in der Klasse `Token` auf den ihnen übergebenen Wert.

3 Grid (0.5 + 1 + 0.5 + 0.5 + 0.25 + 0.5 + 0.5 + 4 = 7.75 Punkte)

Im Unterpaket `model` finden Sie eine Klasse `Grid`, welche das Interface `IGrid` implementiert. Einige der benötigten Methoden in der Klasse sind schon vollständig implementiert, die restlichen Methoden sind nur als Dummy-Implementierung vorhanden und sollen in dieser Aufgabe realisiert werden.

Das `Grid` ist die Darstellung des Spielfelds, auf dem die Felder (Token) platziert sind. (oder vielmehr, dass aus den Feldern besteht). Es speichert die Beziehung zwischen den Feldern, die ausgewählten Token und den bisherigen Highscore.
Sie können das `Grid` als eine Art Matrix betrachten, die 8x18 Token enthält, die die verschiedene Spielsteine repräsentieren. Insgesamt hat das `Grid` jedoch eine Größe von 10x20 Tokens. Es gibt Rand-Tokens um die normalen Tokens, die Sie beim Spielen anklicken, herum, die bestimmte Eigenschaften haben und den Rand des Spielfelds bilden. Diese Rand-Tokens sind im gelösten Zustand, wodurch sie nicht zu sehen sind. Sie dienen dazu, einen Startweg um das Spielfeld zu schaffen, über den die normalen Tokens gelöst werden können.
Die Tokens haben einen Wert (value) und einen Anzeigewert/DisplayValue. Der Anzeigewert ist entweder eine Zahl von 0-9 oder ein Buchstabe von A-Z und wird den Tokens zugewiesen, nachdem sie erfolgreich aus einer Map-Datei gelesen wurden. Die Map-Datei speichert die Werte der Tokens, welche sich von den Anzeigewerten unterscheiden (hierzu mehr in Aufgabe 3h)). Die Rand-Tokens haben den Wert -1 und sind nicht in der Map-Datei enthalten.
Abbildung 2 dient als Beispiel, um den Aufbau des `Grids` zu veranschaulichen. Es zeigt ein kleineres `Grid` mit seinen

einzelnen Feldern. Beachten Sie jedoch, dass das tatsächliche `Grid` im Spiel eine andere Größe und andere Werte hat. Abbildung 2d verdeutlicht, wie auf einzelne Felder im `Grid` zugegriffen werden kann. Der Zugriff erfolgt über die Schreibweise `grid[x][y]`. Wenn Sie beispielsweise auf das rot markierte Token zugreifen möchten, entspricht dies `grid[6][3]`. Beachten Sie, dass die tatsächlichen Koordinaten im Spiel von der Größe und den Werten des `Grid`s abhängen.

(a) `Grid`: Werte der Tokens ohne Rand = 4x9 Matrix

(b) `Grid`: Werte der Tokens mit Rand = 6x11 Matrix

(c) `Grid`: Anzeigewerte der Tokens

(d) `Grid`: Zugriff auf Token = grid[6][3]

Abbildung 2: Beispielhafter Aufbau eines 4x9 (bzw. 6x11) `Grid`s.

a) Deklarieren Sie zusätzlich zu den schon vorhandenen Instanzvariablen die folgenden drei privaten Variablen in der Klasse `Grid`.

- `grid`: Diese Variable soll vom Typ `IToken[][]` und statisch sein. Das 2D-Array soll die Informationen über das Spielfeld selbst halten.

  Hinweis: Felder am Rand des Spielfeldes sollen mit dem Wert `-1` initialisiert werden.

- `selectedTokenOne`. Diese Variable soll vom Typ `IToken` sein und repräsentiert den ersten ausgewählten Token.

- `selectedTokenTwo`. Diese Variable soll vom Typ `IToken` sein repräsentiert den zweiten ausgewählten Token.

Hinweis: Beide `selectedToken`-Variablen sollen direkt bei der Deklaration mit `null` initialisiert werden.

b) Implementieren Sie einen Konstruktor für die Klasse `Grid`, welcher den Dateipfad zu einer Map-Datei als `String` übergeben bekommt. Die Methode erstellt das Grid mit Hilfe der Methode `parseMap(String path)` (siehe 3h)), ruft die vorimplementierte Methode `fillTokenPositions()` auf, die für die Pfadbestimmung benötigt wird und initialisiert die Variable `score` auf einen sinnvollen Anfangswert.

Hinweis: Überlegen Sie bei der Initialisierung der Variable `score`, wo im Projekt schon ein Anfangswert vorgegeben sein könnte.

Hinweis: Die Methode `parseMap(String path)` können Sie in dieser Aufgabe schon verwenden, obwohl diese erst später implementiert wird.

c) Implementieren Sie die öffentliche Methode `IToken getTokenAt(int x, int y)`, welche den Token an der übergebenen Koordinate im Grid zurückgibt. Beachten Sie hierbei unbedingt, dass ungültige Werte abgefangen werden müssen. Sollte ein ungültiger Wert übergeben werden, geben sie `null` zurück.

d) Implementieren Sie die öffentlichen Getter-Methoden `getGrid()` und `getActiveTokens()` für die jeweiligen Datenfelder `grid` und `activeTokens`.

Hinweis: Bitte geben Sie bei dem Getter für die `activeTokens` den Inhalt der beiden Datenfelder `selectedTokenOne` und `selectedTokenTwo` als `IToken`-Array zurück.

e) Implementieren Sie die öffentliche Methode `boolean bothClicked()`, welche zurückgibt, ob zwei Token ausgewählt (selektiert) wurden.

Hinweis: Sie können dies einfach durchführen, indem Sie prüfen, ob die beiden Felder `selectedTokenOne` und `selectedTokenTwo` gesetzt sind. Es muss nicht auf den Status der `Token`s geprüft werden!

f) Implementieren Sie die öffentliche Methode `deselectToken(IToken token)`, welche, vorrausgesetzt der gegebene Token ist selektiert, seinen Status (`TokenState`) auf `DEFAULT` setzt und die Selektierung aufhebt. Implementieren Sie weiterhin die öffentliche Methode `deselectTokens()`, welche beide zuvor selektierten Token deselektiert.

Hinweis: Beide Methoden sollen keinen Rückgabewert besitzen.

g) Implementieren Sie die öffentliche statische Methode `boolean isSolved()`, welche nur dann `true` zurückgibt, wenn alle Felder auf dem Spielfeld gelöst sind (also den `TokenState` `TokenState.SOLVED` besitzen).

h) Studieren Sie das Format einer Map-Datei (*/assets/maps/001.map*).
Implementieren Sie eine private Methode `Token[][] parseMap(String path)`, die den Pfad zu einer Map-Datei erhält und daraus ein 2-dimensionales Array von `Token` konstruiert, welches das Spielfeld darstellen soll. Alle normalen Tokens (also keine Rand-Tokens, siehe Hinweis) haben einen Wert (`value`), welcher in der Map-Datei steht, einen Status (`TokenState`), welcher bei allen normalen Token anfangs `DEFAULT` ist, sowie eine Position auf dem Spielfeld als `Vector2f`. Dabei soll die erste Dimension x und die zweite y repräsentieren. Bedenken Sie, dass Sie mit der Hilfsklasse `IOOperations` den Inhalt von Dateien als String erhalten können und dass einzelne Zeilen in Map-Dateien mit dem String `System.lineSeparator()` separiert sind (auch, wenn Sie die entsprechenden Symbole im Texteditor nicht angezeigt bekommen).

Hinweis: Beachten Sie, dass der Rand des Spielfeldes aus gelösten (`TokenState.SOLVED`) Token besteht, welche nicht in den Map-Dateien enthalten sind! Sie brauchen also keine 8x18 Matrix, sondern eine 10x20 Matrix, welche aus einem Rand an den äußeren Feldern besteht. Dieser Rand kennzeichnet sich dadurch, dass die äußeren Felder den Wert `-1` besitzen.

Hinweis: Die Methode `String.split(String)` könnte für diese Aufgabe von Nutzen sein.

Hinweis: Es ist nicht gewünscht, dass auf diese Methode von außerhalb dieser Klasse zugegriffen werden kann.

4 Highscore (2.5 + 2 = 4.5 Punkte)

Im Paket `model` finden Sie die Klassen `HighscoreEntry` und `Highscore`. Diese stehen jeweils für einen einzelnen Eintrag in der Highscoreliste sowie für den gesamte Highscore. Sobald Sie mit der Implementierung dieser Klassen beginnen, könnten Probleme mit der Highscore-Datei auftreten, da diese fehlerhaft beschrieben worden sein könnte. Sollten daher dort neue Fehler oder Exceptions auftreten, dann könnten Sie zunächst versuchen die Datei *assets/highscore/highscores.hs* zu löschen. Die Datei wird erneut erstellt, wenn ein neuer Highscore erreicht wurde.

4.1 Highscore-Eintrag (0.5 + 0.25 + 0.75 + 0.5 + 0.5 = 2.5 Punkte)

Sehen Sie sich zunächst die Klasse `HighscoreEntry` an. Machen Sie sich zunächst mit der Klasse vertraut und sehen Sie sich insbesondere die Instanzvariablen an, sodass Sie deren Bedeutung kennen. Die Klasse hat drei private Instanzvariablen, die für Sie von Bedeutung sein sollten.

- `LocalDateTime date`: Dieses Feld steht für das Datum und die Uhrzeit des gespielten Spiels.
- `double duration`: Dieses Datenfeld steht für die gebrauchte Zeit in Sekunden.
- `int score`: Dieser Eintrag steht für die erreichte Punktzahl.

Wenn Sie sich Abbildung 1d ansehen, dann würde ein `HighscoreEntry` für einen der Einträge stehen, also z. B. für den ersten Eintrag mit Datum und Uhrzeit 29.05. 15:48, dem erreichten Score von 285 sowie der benötigten Zeit 414 (angezeigt in Sekunden!).

Die Methoden sind in der Klasse bereits mit einer ”Dummyfunktionsweise” vorgegeben. Implementieren Sie nun nachfolgend diese Methoden:

a) Implementieren Sie die Methode `validate`. Diese soll für die übergebenen Parameter folgendes sicherstellen:

- Das Datum darf nicht `null` sein.
- Die Dauer darf nicht negativ sein.
- Der Score darf nicht negativ und nicht größer als 1000 sein.

Ist eine der Bedingungen verletzt, soll von der Methode eine `IllegalArgumentException` geworfen werden.

Hinweis: Bedenken Sie, dass eine `LocalDateTime` auch `null` sein kann und Sie dies abfangen müssen.

b) Implementieren Sie den Konstruktor `public HighscoreEntry(LocalDateTime date, int score, double, duration)`. Dieser soll zunächst prüfen, ob die übergebenen Parameter gültig sind. Ist dies der Fall, sollen die Instanzvariablen mit den entsprechenden Parametern initialisiert werden.

c) Implementieren Sie den zweiten Konstruktor `public HighscoreEntry(String data)`. Dieser bekommt den Highscore-Eintrag als einen `String` im Format ”dd.MM.yyyy HH:mm;score;duration” übergeben. Die einzelnen Einträge sind durch ein ”;” abgetrennt. Der Konstruktor soll prüfen, ob der Parameter gültig ist - also dem genannten Format entspricht und weder `null` noch leer ist - und es soll auch sichergestellt werden, dass alle Einträge vorhanden sind. Ist dies nicht der Fall, soll eine `IllegalArgumentException` geworfen werden.

Sind alle Einträge vorhanden, sollen diese, wie im ersten Konstruktor, validiert werden. Abschließend sollen die Datenfelder entsprechend der zugehörigen Werte initialisiert werden.

- Gültige Parameter: ”04.01.1996 07:07;123;851256.0”, ”23.07.2018 14:23;20;49546.0”, ”23.05.2019 18:20;42;142569.0”.
- Ungültige Parameter: ””, ”iD-12”, ”;7;”, ” ;7;15;1300”, ”5.1.;6;-1”, usw.

Hinweis: Sie können zum Aufteilen des `String`s die Methode `split()` der Klasse `String` nutzen! Eine andere Möglichkeit wäre die Implementierung mit einer Schleife. Außerdem benötigen Sie die Methode `parse()` der Klasse `LocalDateTime`.

d) Schreiben Sie die Methode `public boolean equals(Object obj)`. Diese vergleicht den aktuellen `HighscoreEntry` mit dem übergebenen Parameter. Es soll also zunächst geprüft werden, dass der Parameter nicht `null` und vom Typ `HighscoreEntry` ist. Ist dies der Fall, sollen die Werte der Datenfelder verglichen werden. Nur wenn abschließend alle Datenfelder gleich sind, soll die Methode `true` zurückliefern. In allen anderen Fällen ist die Rückgabe `false`.

e) Implementieren Sie die Methode `compareTo(HighscoreEntry o)`. Die Methode wird vom Spiel dazu genutzt die Highscore-Einträge richtig zu sortieren. Die Methode vergleicht die Werte der Highscore-Einträge und hat folgende Rückgaben:

1. Ist der Score des aktuellen `HighscoreEntry` kleiner als der Score des übergebenen `HighscoreEntry`, soll die Rückgabe eine beliebige positive Zahl sein.
2. Ist der Score des aktuellen `HighscoreEntry` größer als der Score des übergebenen `HighscoreEntry`, soll die Rückgabe eine beliebige negative Zahl sein.
3. Ist der Score gleich, wird die Dauer invers zu 1. und 2. verglichen. Dies bedeutet, dass eine kürzere Dauer als dominierend gewertet werden soll. Ist aber abschließend die Dauer gleich, soll die Rückgabe 0 sein.

4.2 Highscore (1 + 1 Punkte)

Abschließend zum Highscore müssen noch zwei Methoden in der Klasse `Highscore` implementiert werden. Diese sind dafür zuständig, dass erst Elemente hinzugefügt und auch abschließend in der Datei *highscores.hs* gespeichert werden.

a) Implementieren Sie die Methode `addEntry(HighscoreEntry entry)`, welche der passenden Highscore Liste den neuen Eintrag hinzufügt, wenn dieser noch in die Liste passt. Ist in der Liste kein Platz mehr (bereits zehn Einträge vorhanden) und der neue Highscore ist höchstens genauso gut wie der bereits letzte existierende Eintrag, dann soll der neue Eintrag nicht zur Highscore-Liste hinzugefügt werden. Ansonsten soll der Eintrag hinzugefügt und die Highscore-Einträge sortiert werden, sodass eine Sortierung wie in Abbildung 1d zustande kommt. Die Liste soll also wieder zuerst nach Score und erst bei gleichem Score nach Dauer des jeweiligen Spiels sortiert werden. Am Ende soll die Liste aber wieder maximal aus den zehn besten Einträgen bestehen.

   Hinweis: Holen Sie sich zunächst die passende Liste und sehen Sie, wie viele Elemente bereits in der Liste vorhanden sind.

   Hinweis: Die Klasse `Collections` besitzt Methoden, die bei der Bearbeitung dieser Aufgabe von Nutzen sein können.

b) Abschließend soll die Methode `saveToFile(String fileName)` implementiert werden. Diese ist dafür zuständig alle Highscore-Einträge (aus der Liste) in die Highscore-Datei zu schreiben. Dazu sollen zunächst alle Einträge in einem `String` zusammengefasst werden, wobei jeder Eintrag in eine neue Zeile geschrieben werden soll. Zum Schluss wird der `String` in besagte Datei geschrieben werden.

   Hinweis: Nutzen Sie die Methode `System.lineSeparator()`, um in einem `String` eine neue Zeile zu beginnen.

   Hinweis: Zum Erstellen der Datei könnte eine Methode aus der Klasse `IOOperations` hilfreich sein.

   Hinweis: Sie sollten eventuelle Exceptions der Collection `highscore` abfangen.

   Haben Sie alles richtig gemacht, sollte die Highscore-Datei wie in Listing 1 aufgebaut sein.

Listing 1: Beispielhafte Einträge in der "highscores.hs"-Datei.

```
07.10.2018 07:40;205;6999.0
20.05.2019 18:35;110;3123.0
08.03.2019 22:13;20;4875.0
```

5 Cheats (1 + 0.5 + 3 + 3 + 3 = 10.5 Punkte)

In dieser Aufgabe müssen die Übergabeparameter nicht überprüft werden. Zudem müssen für jeden Cheat bei erfolgreicher Ausführung die Kosten des Cheats vom Konto des Spielers abgezogen werden. Erfolgreiche Ausführung heißt in diesem Fall, dass beim Ausführen des Cheats genug Punkte vorhanden sind. Zur Preisermittlung kann die Methode `getCheatCost()` genutzt werden.

Hinweis: Ein Cheat soll auch Punkte vom Score abziehen, wenn er keine sinnvolle Lösung bringt. (Also z.B. kein valides Paar an Feldern mehr vorhanden ist.)

Hinweis: Die Klasse ist erst sinnvoll nutzbar, wenn alle Teilaufgaben implementiert wurden. Alle Cheat-IDs sowie Cheatkosten sind in der `Consts`-Klasse im `util`-Paket zu finden.

Hinweis: Beachten Sie, dass Cheats erst dann ausgeführt werden können, wenn Sie genug Punkte im Spiel erzielt haben. Um das Fehlersuchen einfacher zu machen, können Sie in der Klasse `Consts` die Konstante `START_POINTS` auf einen höheren Wert setzen, sodass die Cheats aktivierbar sind. Vergessen Sie jedoch nicht, den Wert vor Abgabe des Projekts wieder auf `0` zu setzen!

a) Schreiben Sie die private statische Methode `boolean isCheatPossible(final Grid grid, final int cheatID)`. Diese soll als Wahrheitswert zurückgeben, ob der übergebene Cheat im übergebenen `Grid` derzeit möglich ist. Nutzen Sie dazu die Werte aus der `Consts`-Klasse.

   Hinweis: An dieser Stelle ist gefordert, dass die Möglichkeit der Cheat-Ausführung gegeben ist, sofern der aktuelle Score groß genug ist. Es ist nicht notwendig das Grid auf valide Feldpaare zu prüfen!

b) Als Nächstes soll die private statische Methode `List<IToken> shuffle(List<IToken> list)` implementiert werden. Diese soll eine übergebene Liste von `IToken`s zufällig durchmischen und sie dann wieder zurückgeben.

c) Implementieren Sie dann die private statische Methode `List<IToken> findTokensWithType(final IToken token, final Grid grid)`. Im `Grid` soll nach allen `IToken`s gesucht werden, die den gleichen Typ (Anzeigewert/DisplayValue) wie das übergebene `IToken` besitzen. Alle gefundenen `IToken`s sollen in Form einer gemischten `ArrayList` zurück gegeben werden. Zum Mischen kann die Methode `shuffle()` verwendet werden.

   Hinweis: Beachten Sie, dass der übergebene `IToken` selbst **nicht** in die Liste eingefügt werden soll. Auch bereits gelöste `IToken`s sollen nicht in die Rückgabeliste eingefügt werden.

d) Schreiben Sie die öffentliche, statische Methode `void solvePair(final Grid grid)`. Der Cheat soll ein derzeit lösbares Paar im übergebenen `Grid` finden und es für den*die Spieler*in lösen. Um ein lösbares Paar zu finden, können Sie die statische Methode `findValidTokens(final Grid grid)` nutzen. Wählen Sie mithilfe einer `Grid`-Methode alle derzeit ausgewählten `Token`s ab und wählen stattdessen durch eine weitere `Grid`-Methode die gefundenen `IToken`s an. Als letzten Schritt sollten Sie dann den Score des*der Spieler*in im `Grid` um den entsprechenden Wert verringern, wobei bei einem passenden Felderpaar **nicht** erneut fünf Punkte für das Lösen vergeben werden sollen. (Es wird also nur die Punktzahl für den Cheat abgezogen.)

   Hinweis: Diese Methode soll also, sofern sie Dank eines ausreichenden Punktestandes ausgeführt werden kann, immer genau 20 Punkte abziehen (auch wenn kein Paar gelöst wird).

e) Implementieren Sie nun die öffentliche und statische Methode `void findPartner(final Grid grid)`. Dieser Cheat sucht für eine angeklickte Karte einen lösbaren Partner. Falls kein lösbarer Partner gefunden wurde, wird ein zufälliger Partner mit dem gleichen Typ auf dem `Grid` ausgewählt.

   Gehen Sie dabei wie folgt vor:
   Prüfen Sie zunächst, ob derzeit genau ein `IToken` im `Grid` ausgewählt ist. Falls dies nicht der Fall ist, soll die Methode beendet werden. Nutzen Sie dann Ihre Implementierung von `findTokensWithType(final IToken token, final Grid grid)`, um alle `IToken`s im `Grid` zu finden, welche als Partner für das ausgewählte `IToken` in Frage kommen. Prüfen Sie alle dieser möglichen Partner mit der vorimplementierten statischen Methode `solvable(IToken token1, IToken token2, Grid grid)`. Die Methode liefert genau dann `true` zurück, wenn die übergebenen `IToken`s im übergebenen `Grid` lösbar sind. Wenn ein valides Paar gefunden wurde, kann der Partner im `Grid` ausgewählt und der Score entsprechend verringert werden. (An dieser Stelle werden dann keine Punkte für das korrekte Lösen des Paares vergeben.) Wird keine lösbare Kombination gefunden, soll eine nicht lösbare Kombination im `Grid` gewählt und der Score ebenfalls um die Cheatkosten **und** die Kosten für eine falsche Paarauswahl verringert werden.

   Hinweis: Die Methode soll also, vorausgesetzt zu Beginn ist ein groß genuger Score vorhanden, entweder 10 oder 15 Punkte abziehen (je nachdem, ob ein lösbarer Partner gefunden wurde).

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: Sie können deutsch oder englisch kommentieren – entscheiden Sie sich für eine Variante!

7 Testen

Testen Sie Ihre Methoden ausgiebig mit den in Vorlesung und Übung vorgestellten Techniken. Prüfen Sie die Methoden mit invaliden Parametern und decken Sie etwaige Sonderfälle ab.

Wir wünschen Ihnen viel Spaß und Erfolg!