2048 (15 Punkte)

In diesem Projekt werden Sie das beliebte Spiel 2048 [1, 2] in Java implementieren.

1 Das Spiel

2048 ist ein Denkspiel, bei dem man Steine auf dem Spielfeld verschieben muss. Ziel ist es, Steine mit immer höheren Werten bis hin zur 2048 zu erzeugen. Das Spiel beginnt mit zwei Steinen, die entweder den Wert 2 oder 4 haben. In jeder Runde darf der Spieler eine Richtung wählen, in die alle Steine – falls möglich – geschoben werden. Ein Stein kann nur verschoben werden, falls das Nachbarfeld in der gewählten Richtung frei ist. Falls auf dem nächsten belegten Feld in dieser Richtung ein Stein des gleichen Wertes ist, werden beide entfernt und ein neuer Stein entsteht auf diesem Feld mit dem summierten Wert der beiden alten. Nach jeder Runde wird ein neuer Stein zufällig auf dem Spielfeld plaziert. Wie auch die initialen Steine hat er mit einer Wahrscheinlichkeit von 90% den Wert 2 und mit einer Wahrscheinlichkeit von 10% den Wert 4. Das Spiel ist verloren, sobald es keine Richtung mehr gibt in die man die Steine auf dem Spielfeld verschieben kann. Dies ist der Fall, wenn alle Felder belegt sind und es keine zwei Steine des gleichen Wertes gibt, die nebeneinander liegen. In allen anderen Fällen kann man entweder Steine in die noch freien Felder schieben oder benachbarte Steine des gleichen Wertes zu einem neuen Stein verschmelzen und damit ein freies Feld gewinnen.

Abbildung 1: Grafische Benutzerschnittstelle

2 Die Aufgaben

In diesem Projekt gibt es zwei konzeptionell unterschiedliche Aufgaben: Testerstellung und Implementierung. Die Implementierung des Projekts kann über den gesamten Projektzeitraum erfolgen, die Testerstellung jedoch nur in der ersten Woche. In diesem Zeitraum sind Sie dazu angehalten, nicht nur an der Implementierung Ihres Projektes zu arbeiten, sondern auch selbst Tests zu schreiben. Diese sollten dazu dienen die Spezifikationen aus Tabelle 1 zu überprüfen. In diesem Sinne führen wir Ihre Tests mit sowohl fehlerhaften als auch fehlerfreien Implementierungen des Simulators aus und erwarten, dass Ihre Tests bei genau den ersteren fehlschlagen. Die Implementierung besteht wiederum aus drei Teilen: dem Simulator, der die Spiellogik enthählt, der Benutzerschnittstelle, die erlaubt das Spiel zu steuern, und einem Computerspieler (Bonus!) der vollautomatisch spielt. Die einzelnen Teile werden nun im Detail beschrieben.

2.1 Der Simulator (6 Punkte)

Im Simulator ist die Spiellogik implementiert, er kontrolliert und steuert also den Ablauf des Spiels. Zu Beginn des Spiels wird das Spielfeld initialisiert, d.h. zwei zufällige Steine werden platziert und die erste Runde beginnt. Jede Runde verläuft nach dem gleichen Prinzip: Der Spieler wählt eine gültige Richtung, in die die Steine geschoben werden, und ein neuer Stein wird platziert. Dies geschieht solange, bis das Spielfeld komplett mit Steinen belegt ist

und keine zwei des gleichen Wertes nebeneinander liegen. Ist dies der Fall wird keine neue Runde mehr gestartet, sondern das Spiel mit einer Nachricht (“Game Over”) beendet. Diese Nachricht enthält auch die finale Punktzahl sowie die Anzahl der gespielten Runden (vgl. Abbildung 2).
Die Regeln zum Bewegen und Verschmelzen der Steine sind die Gleichen wie in der online Variante des Spiels [1]. Zusätzlich sind in Abbildung 4 die interessantesten Fälle grafisch erklärt. Punkte werden nur beim Verschmelzen vergeben und entsprechen dem Wert des neu entstanden Steins.

Abbildung 2: “Game Over” Benachrichtigung am Ende eines Spiels.

Technische Hinweise
In der Datei `SimulatorInterface.java` finden Sie die *dokumentierte* Simulatorschnittstelle, die Sie (z.B. in einer `Simulator.java`) implementieren sollen. Auch die zur Testerstellung benötigten Spezifikationen sind dort in Form von JavaDoc-Kommentaren enthalten. Da Sie für diese Spezifikationen Tests generieren sollen, gibt es keine öffentlichen Tests für die betroffenen Methoden. Es bietet sich deshalb an, erst die Tests (vgl. Abschnitt 2.3) zu erstellen, bevor Sie mit der Implementierung der betroffenen Methoden beginnen. Beachten Sie hierbei auch das Vorbedingungen jeglicher Art durch *Zusicherungen (assertions)* abgesichert werden müssen.
Um das Spiel spielen zu können, benötigen Sie einen (menschlichen oder Computer-) Spieler der die entsprechende Schnittstelle in der Datei `PlayerInterface.java` implementiert. Der menschliche Spieler soll dabei über die Benutzerschnittstelle einen Zug auswählen, der Computerspieler kann anhand des aktuellen Zustands des Spielfelds im Simulator einen geeigneten Zug berechnen.
Bitte beachten Sie, dass Sie nur das Java-Paket `ttfe` und Unterpakete für Ihre Implementierung nutzen. Weitere Pakete werden von der Testinfrastruktur nicht berücksichtigt.

2.2 Der Computerspieler (Bonus: 5 Punkte)

Zum Abschluss des Projekts sollen Sie einen Computerspieler schreiben, der statt eines Menschen versucht so viele Spiele wie möglich zu “gewinnen”. Genau wie ein Mensch kann Ihr Computerspieler den momentanen Zustand des Spielfelds begutachten und je nach Komplexität Ihrer Implementierung auch die möglichen folgenden Runden berücksichtigen.
Wie viele Punkte Sie für Ihren Computerspieler bekommen ist abhängig davon wie weit er im Durchschnitt kommt. Wenn ihr Computerspieler in der Hälfte der Spiele einen 2048 Stein erreicht, bekommen Sie alle Bonuspunkte.
Während beim Simulator die Dimensionen des Spielfelds nicht festgelegt sind, reicht es wenn Ihr Computerspieler auf einem 4x4 Spielfeld funktioniert. Sie können in Ihrem Computerspieler beliebige Methoden auf dem Simulator Objekt aufrufen, insbesondere auch Methoden die den Zustand des Spielfelds verändern. Allerdings wird, bevor der

vom Computerspieler ausgewählte Zug ausgeführt wird, der vorherige Zustand des Simulators wiederhergestellt. Beachten Sie auch, dass der Simulator den neuen Stein nicht unbedingt auf dem gleichen Feld erzeugt wie wenn Sie seine `performMove` Methode in Ihrem Computerspieler aufrufen.
Ein Spiel wird nach maximal 10 Sekunden[1] automatisch für beendet erklärt.

2.2.1 Wettbewerb

Um Ihnen einen besonderen Anreiz zu geben, einen möglichst guten Computerspieler zu schreiben, werden wir am Ende des Semesters einen Wettbewerb abhalten. Hierbei gelten folgende Regeln:

- Das Spiel wird nicht automatisch nach 10 Sekunden beendet, allerdings hat Ihr Computerspieler jeweils maximal 1 Sekunde Zeit um den nächsten Zug auszugeben.
- Beim Verschmelzen von Steinen wird nicht nur der Wert x des neuen Steins zur Punktzahl addiert, sondern $x * 2^{v-1}$, wobei v die Anzahl der gleichzeitig verschmolzenen Steine ist. Zum Beispiel würde es beim Verschmelzen der vier 2er Steine in Abbildung 4 zu zwei 4er Steinen 16 statt normal 8 Punkten geben. Falls gleichzeitig noch zwei weitere 2er Steine zu einem 4er Stein verschmolzen wären, also insgesamt 6 2er Steine zu 3 4er Steinen, gäbe es insgesamt $3 * 4 * 2^2 = 48$ Punkte.

Die besten Implementierungen werden in einem Turniersystem gegeneinander antreten bis ein Sieger feststeht. Die Teilnahme am Wettbewerb ist optional und gibt keine weiteren Bonuspunkte.

Teilnahme Für die Teilnahme am Wettbewerb müssen Sie ihr Projekt noch einmal in einem zweiten git Depot abgeben, getrennt von der restlichen Abgabe des Projekts. Das git Depot finden Sie hier:

```
git clone https://prog2scm.cdl.uni-saarland.de/git/competition/$NAME
```

wobei Sie `$NAME` durch ihren Benutzernamen ersetzen müssen. Das Depot wird im Laufe des Projekts freigeschaltet. Sie können Ihre 2048 Implementierung dorthin kopieren. Ihren Computerspieler für den Wettbewerb können Sie in diesem Depot einreichen, mit git push bis zum **1.7.2018, 23:59**.

2.3 Testerstellung (9 Punkte)

Nach der ersten Woche werden die von Ihnen geschriebenen Tests bewertet. Um ihre Punkte zu bestimmen, werden wir Ihre Tests nutzen, um verschiedene, teilweise fehlerhafte, Simulatorimplementierungen zu testen. Sie erhalten Punkte, wenn Ihre Tests die Implementierung korrekt als fehlerhaft oder fehlerfrei erkennen. Das bedeutet, dass auf fehlerhaften Implementierungen mindestens einer ihrer Tests fehlschlägt, und auf fehlerfreien alle Tests korrekt ausgeführt werden. Um zu definieren wie sich eine Implementierung verhalten soll, haben wir in Tabelle 1 Spezifikationen definiert. Falls eine dieser Spezifikationen nicht eingehalten wird, gilt eine Implementierung als fehlerhaft.

Technische Hinweise
Alle Tests die zur Bewertung dieser Aufgabe herangezogen werden sollen, müssen Sie in dem Java-Paket ttfe.tests implementieren. Es ist unerheblich, ob Sie dazu mehrere Klassen benutzen, oder nicht.
Wenn Sie Hilfsmethoden/-klassen für Ihre Tests implementieren möchten, müssen diese ebenfalls in diesem Paket implementiert sein. Ihren Tests stehen außerdem nur die zu Anfang vorgegebenen Klassen und Schnittstellen, und deren Methoden, aus dem Paket ttfe zur Verfügung. Deren Implementierung, insbesondere die der `TTFEFactory`, werden von uns gestellt.
Beachten Sie, dass wir überprüfen, dass Ihre Tests das Verhalten des Simulators so testen, dass ein `AssertionError` (oder eine Unterklasse) geworfen wird.
Außerdem dürfen Sie, da die Testerstellung für den Simulator bewertet wird, hierfür keine Tests von Kommilitonen benutzen, wie dies in bisherigen Projekten der Fall war.

[1] Die Ausführung erfolgt auf unserer Hardware und ohne eine Benutzerschnittstelle.

Methodenname	Spezifikation
`addPiece`	Positioniert die neuen Steine zufällig und wählt den Wert zufällig aber mit den gegebenen Wahrscheinlichkeiten aus. Vorbedingung ist ein nicht komplett belegtes Spielfeld.
`getBoardWidth`	Der Rückgabewert ist nicht negativ und wie beim Erstellen des Spiels spezifiziert.
`getBoardHeight`	Der Rückgabewert ist nicht negativ und wie beim Erstellen des Spiels spezifiziert.
`getNumMoves`	Der Rückgabewert ist nicht negativ und gibt die Anzahl der gemachten Züge an.
`getNumPieces`	Der Rückgabewert ist nicht negativ und gibt die Anzahl der momentan auf dem Spielfeld befindlichen Steine wieder.
`getPoints`	Der Rückgabewert ist nicht negativ und gibt die Anzahl der erreichten Punkte an.
`getPieceAt`	Liefert den Wert des Steins an den gegebenen Koordinaten (vgl. Abbildung 3) zurück, bzw. 0 falls dort kein Stein liegt. Vorbedingung sind gültige Koordinaten.
`setPieceAt`	Setzt einen Stein an die gegebenen Koordinaten mit dem gegebene Wert. Vorbedingung sind gültige Koordinaten.
`isMovePossible`	Gibt genau dann wahr (`true`) zurück, falls es eine mögliche Zugrichtung gibt, bzw. die gegebene Richtung gültig ist.
`isSpaceLeft`	Gibt genau dann wahr (`true`) zurück, falls es mindestens ein freies Feld gibt.
`performMove`	Versucht einen Zug in die gegeben Richtung auszuführen und gibt genau dann wahr (`true`) zurück, falls dies erfolgreich war.

Tabelle 1: Spezifikationen für das Simulator Interface. *Verbindlich* sind auch die im initialen Code angegebenen JavaDoc-Kommentare.

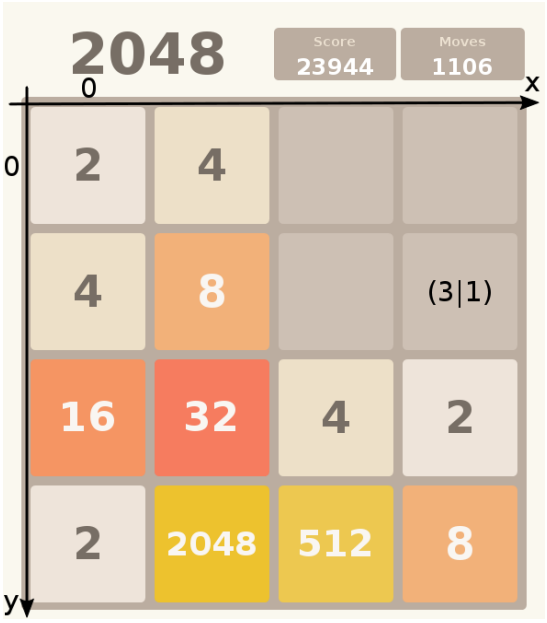

Abbildung 3: Koordinatensystem des Simulators

3 Abgabe

Die Abgabe dieses Projekts ist zweigeteilt. Ihre Tests müssen nach einer Woche, der Simulator und der Computerspieler nach zwei Wochen abgegeben werden. Damit ergibt sich:

Abgabe Tests
Ausschließlich im Java-Paket ttfe.tests. git push bis zum **5.06.2018, 23:59**.

Abgabe Implementierung
git push bis zum **12.06.2018, 23:59**.

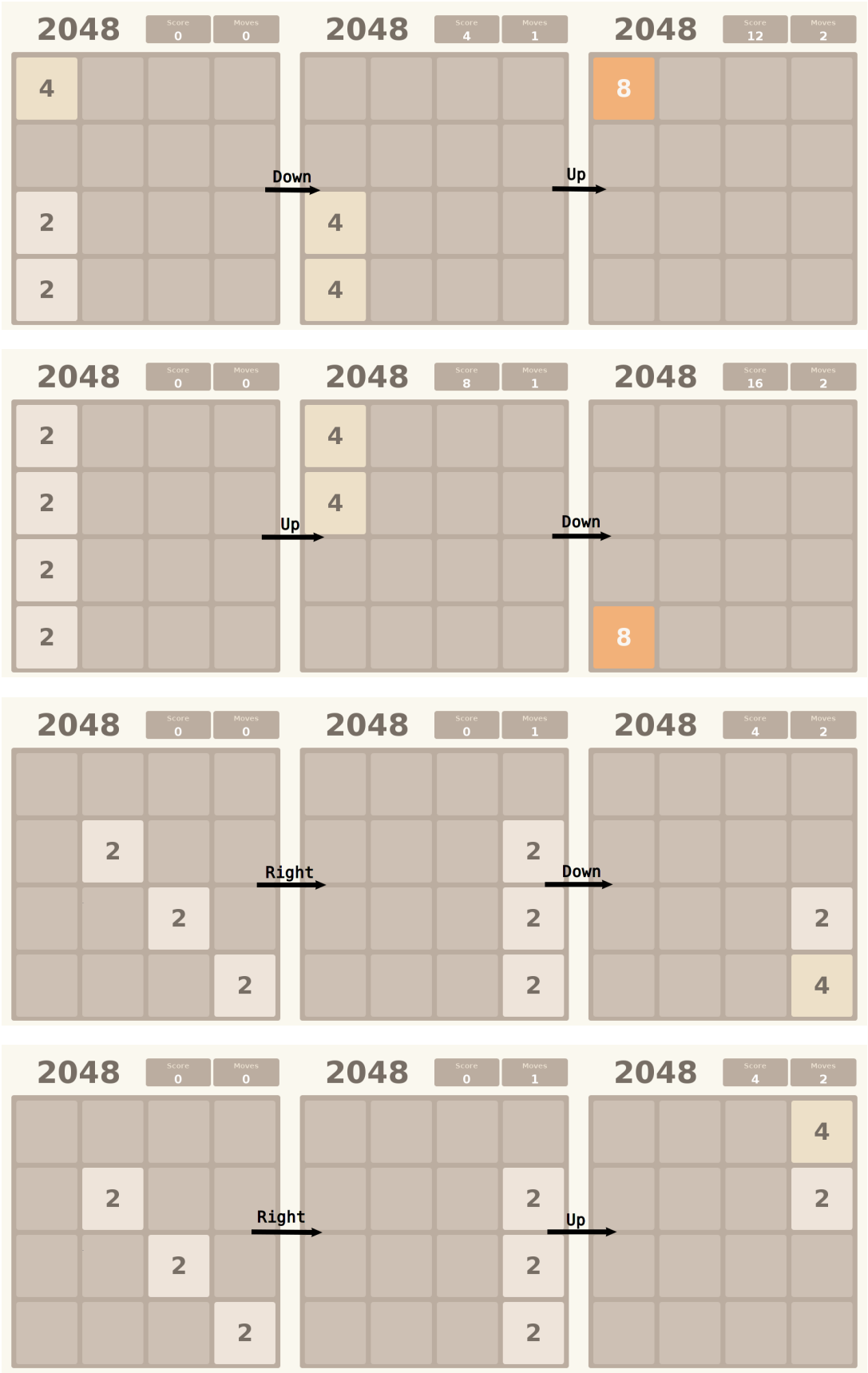

Abbildung 4: Bewegung und Verschmelzung in Beispielsituationen. Bitte beachten Sie, dass wir zur Veranschaulichung keine neuen Steine hinzugefügt haben.

4 Java Projekte – Eclipse

Dieses und die beiden verbleibenden Projekte werden von Ihnen in Java implementiert. Um Ihnen dies zu erleichtern dürfen Sie die Entwicklungsumgebung *Eclipse* [3, 4] benutzen, die bereits auf Ihrer VM installiert ist. Um das Projekt in Eclipse bearbeiten zu können müssen Sie es erst importieren. Öffnen Sie dazu erst Eclipse über das Icon auf dem Desktop. Wenn Sie nach einem "Workspace" gefragt werden übernehmen Sie den Standardpfad `/home/prog2/workspace/` und aktivieren Sie die "Use this as the default ..." Box bevor Sie mit `OK` bestätigen. Die folgenden Schritte müssen Sie für jedes Projekt erneut ausführen:

1. Klonen Sie das Projekt in einen beliebigen Ordner:
   ```
   git clone https://prog2scm.cdl.uni-saarland.de/git/project4/$NAME
   ```
   wobei Sie `$NAME` durch ihren Benutzernamen ersetzen müssen.

2. Benutzen Sie *Import*, ein Unterpunkt des des *File* Menüeintrags in Eclipse, um den Importierdialog zu öffnen.

3. Wählen Sie "Existing project into workspace" aus und benutzen Sie den neuen Dialog um den Ordner des Projekts (siehe Punkt 1) auszuwählen.

Literatur

[1] `http://gabrielecirulli.github.io/2048/`

[2] `http://de.wikipedia.org/wiki/2048_%28Computerspiel%29`

[3] `http://de.wikipedia.org/wiki/Eclipse_(IDE)`

[4] `https://github.com/pellaton/eclipse-cheatsheet`