Programmierung 2 (SS 2021)
Universität des Saarlandes
Fakultät MI
Compiler Design Lab

Projekt 4

Prof. Dr. Sebastian Hack
Julian Rosemann, B. Sc.
Thorsten Klößner, M. Sc.
Julia Wichlacz, M. Sc.
Maximilian Fickert, M. Sc.

2048 (15 Punkte)

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

```
git clone https://prog2scm.cdl.uni-saarland.de/git/project4/$NAME
```

wobei Sie `$NAME` durch ihren Benutzernamen ersetzen müssen.

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 an zufälligen Positionen, 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 doppelten Wert der beiden verschmolzenen Steine. Nach jeder Runde wird ein neuer Stein zufällig auf einem freien Feld des Spielfelds platziert. 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 Spezifikation des Simulators 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 enthält
- Der Benutzerschnittstelle, die erlaubt das Spiel zu steuern (bereits implementiert) und
- Einem Computerspieler der vollautomatisch spielt (Bonus)

Die einzelnen Teile werden nun im Detail beschrieben.

2.1 Der Simulator (2 Wochen – 6 Punkte)

Im Simulator ist die Spiellogik implementiert, er kontrolliert und steuert also den Ablauf des Spiels. Dieser ist von Ihnen selbst zu implementieren. Zu Beginn des Spiels wird das Spielfeld initialisiert, d.h. zwei zufällige Steine werden an zufälligen Stellen platziert und die erste Runde beginnt. Jede Runde verläuft nach dem gleichen Prinzip: Der Spieler wählt eine gültige Richtung, in welche 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 der durch Verschmelzung neu entstandenen Steine.

Technische Hinweise In der Datei `SimulatorInterface.java` finden Sie die *dokumentierte* Simulatorschnittstelle, die Sie durch eine eigene Unterklasse implementieren sollen. Auch die zur Implementierung und Testerstellung benötigten Spezifikationen (siehe Tabelle 1) sind dort in Form von JavaDoc-Kommentaren enthalten. Da Sie für diese Spezifikationen selbst Tests anlegen sollen, gibt es keine öffentlichen Tests für die betroffenen Methoden. Es bietet sich deshalb an, erst die Tests (vgl. Abschnitt 2.2) zu erstellen, bevor Sie mit der Implementierung der betroffenen Methoden beginnen. Beachten Sie hierbei auch, dass Vorbedingungen jeglicher Art geprüft werden müssen: Sollte die Vorbedingung einer Methode beim Aufruf verletzt sein, so muss die in Tabelle 1 spezifizierte Ausnahme geworfen werden.

Eine Instanz des Simulators erhalten Sie über die statische Methode `createSimulator` der Klasse `TTFEFactory`. Implementieren Sie diese Methode so, dass sie eine Instanz Ihrer Implementierung zurückgibt.

Bitte beachten Sie, dass Sie nur das Java-Paket `ttfe` und Unterpakete für Ihre Implementierung nutzen. Weitere Pakete werden von der Testinfrastruktur nicht berücksichtigt. Beachten Sie auch, dass Sie die vorgegebenen Schnittstellen weder verändern noch erweitern dürfen, da diese von unserer Testinfrastruktur benutzt werden.

Abbildung 2: "Game Over" Benachrichtigung am Ende eines Spiels.

2.2 Testerstellung (1 Woche – 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 korrekterweise als fehlerhaft oder fehlerfrei erkennen. Das bedeutet, dass auf fehlerhaften Implementierungen mindestens einer ihrer Tests fehlschlägt, und auf fehlerfreien Implementierungen 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, ansonsten als korrekt.

Einschränkungen Die Bewertung Ihrer Tests unterliegt gewissen Einschränkungen:

- Ihre Tests müssen den Simulator nur mit einer Breite und Höhe des Spielfeldes von 4 instanziieren.
- Ihre Tests müssen nicht überprüfen, ob Zufallseffekte tatsächlich zufällig oder deterministisch erfolgen.
- Ihre Tests müssen nicht überprüfen, ob die Vorbedingungen vom Simulator sichergestellt werden.
- Hinsichtlich der `run`-Methode müssen Ihre Tests lediglich sicherstellen, dass das Spiel in einem Zustand endet, in dem kein Zug mehr möglich ist.

Diese Einschränkungen gelten ausschließlich für die Bewertung Ihrer Tests. Unsere eigenen Tests werden Ihre Simulatorimplementierung auch außerhalb dieser Einschränkungen testen, insbesondere auf das Verhalten bei Nichteinhaltung von Vorbedingungen. Es ist insofern sinnvoll auch Tests zu schreiben, welche Ihren Simulator jenseits dieser Einschränkungen auf Korrektheit überprüfen.

Technische Hinweise Alle Tests die zur Bewertung dieser Aufgabe herangezogen werden, müssen Sie in dem Java-Paket `ttfe.tests` implementieren (ohne weitere Unterpakete). 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. Sie erhalten eine Instanz der zu testenden Schnittstelle über die Methode `TTFEFactory.createSimulator`.

Stellen Sie sicher dass Ihre Tests das Verhalten des Simulators so testen, dass ein `AssertionError` (oder eine Unterklasse) geworfen wird, falls die Implementierung als fehlerhaft erkannt wurde. Da die Testerstellung bewertet wird, dürfen Sie keine Tests von Kommilitonen benutzen, wie dies in bisherigen Projekten der Fall war.

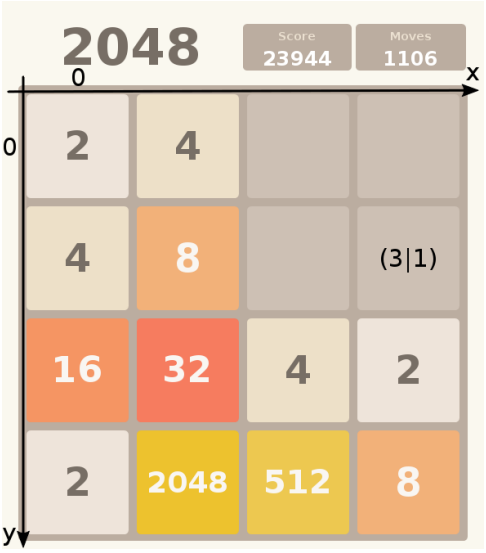

Abbildung 3: Koordinatensystem des Simulators

Methodenname	Spezifikation	Vorbedingung (`Exception`)
Konstruktor	Konstruiert eine neue Instanz des Simulators, welche den Initialzustand des Spiels repräsentiert. Zwei Steine mit gültigem Wert sind also bereits beliebig auf dem Spielfeld platziert.	Spielfeldbreite und -höhe von mindestens 2, Zufallszahlen-generator ist nicht `null`. (`IllegalArgumentException`)
`addPiece`	Positioniert den neuen Stein zufällig auf ein freies Feld und wählt den Wert zufällig wie oben beschrieben aus.	Das Spielfeld ist nicht komplett belegt. (`IllegalStateException`)
`getBoardWidth`	Gibt die Breite des Spielfelds wie beim Erstellen des Spiels spezifiziert zurück.	-
`getBoardHeight`	Gibt die Höhe des Spielfelds wie beim Erstellen des Spiels spezifiziert zurück.	-
`getNumMoves`	Gibt die Anzahl der bisher gemachten Züge zurück.	-
`getNumPieces`	Gibt die Anzahl der momentan auf dem Spielfeld befindlichen Steine zurück.	-
`getPoints`	Gibt die bisher erreichte Punktzahl an.	-
`getPieceAt`	Gibt den Wert des Steins an den gegebenen Koordinaten (vgl. Abbildung 3) zurück, bzw. 0 falls dort kein Stein liegt.	Gültige Koordinaten. (`IllegalArgumentException`)
`setPieceAt`	Setzt einen Stein des spezifizierten Werts an die gegebenen Koordinaten. Ein bereits dort vorhandener Stein wird dabei überschrieben. Wird jedoch der Wert 0 angegeben, so wird stattdessen der Stein an dieser Koordinate entfernt, falls existent.	Gültige Koordinaten, Wert nicht negativ. (`IllegalArgumentException`)
`isMovePossible`	Gibt genau dann wahr (`true`) zurück, falls es eine mögliche Zugrichtung gibt, bzw. die gegebene Richtung gültig ist.	Die angegebene Richtung ist nicht `NULL`. (`IllegalArgumentException`)
`isSpaceLeft`	Gibt genau dann wahr (`true`) zurück, falls es mindestens ein freies Feld gibt.	-
`performMove`	Versucht einen Zug in die angegebene Richtung auszuführen und gibt genau dann wahr (`true`) zurück, falls dies erfolgreich war. Erhöht gegebenenfalls die bisher erreichte Punktzahl.	Die angegebene Richtung ist nicht `NULL`. (`IllegalArgumentException`)
`run`	Startet das Spiel und kommuniziert dessen Ablauf über das UserInterface. In jedem Zustand des Spiels muss dabei der vom Spieler für diesen Zustand angegebene Zug ausgeführt werden. Der folgende Zustand muss mit diesem Zug konistent sein. Im Endzustand dürfen keine Züge mehr möglich sein.	Spieler und UserInterface sind nicht `NULL`. (`IllegalArgumentException`)

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

2.3 Der Computerspieler (2 Wochen – 5 Bonuspunkte)

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. Zu dem gegeben Zustand soll Ihr Computerspieler einen (hoffentlich gültigen) Zug zurückgeben.

Während beim Simulator die Dimensionen des Spielfelds nicht festgelegt sind, reicht es wenn Ihr Computerspieler auf einem 4x4 Spielfeld funktioniert. Ebenfalls läuft das Spiel unter einer Zeitbegrenzung: Ein Spiel wird nach maximal 10 Sekunden[1] automatisch für beendet erklärt. Die bis dahin erreichten Punkte erhalten Sie. Wie viele Projektpunkte 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.

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 `addPiece` Methode in Ihrem Computerspieler aufrufen, da Steine immer an zufälligen freien Stellen generiert werden.

Implementieren Sie den Computerspieler indem Sie die Spielerschnittstelle aus `PlayerInterface.java` mit Ihrer eigenen Unterklasse implementieren. Der Computerspieler wird anschließend über die Methode `createPlayer` der Klasse `TTFEFactory` angefordert. Implementieren Sie diese Methode sodass sie eine Instanz Ihres Computerspielers zurückgibt.

2.3.1 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 **6.7.2021, 23:59**.

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

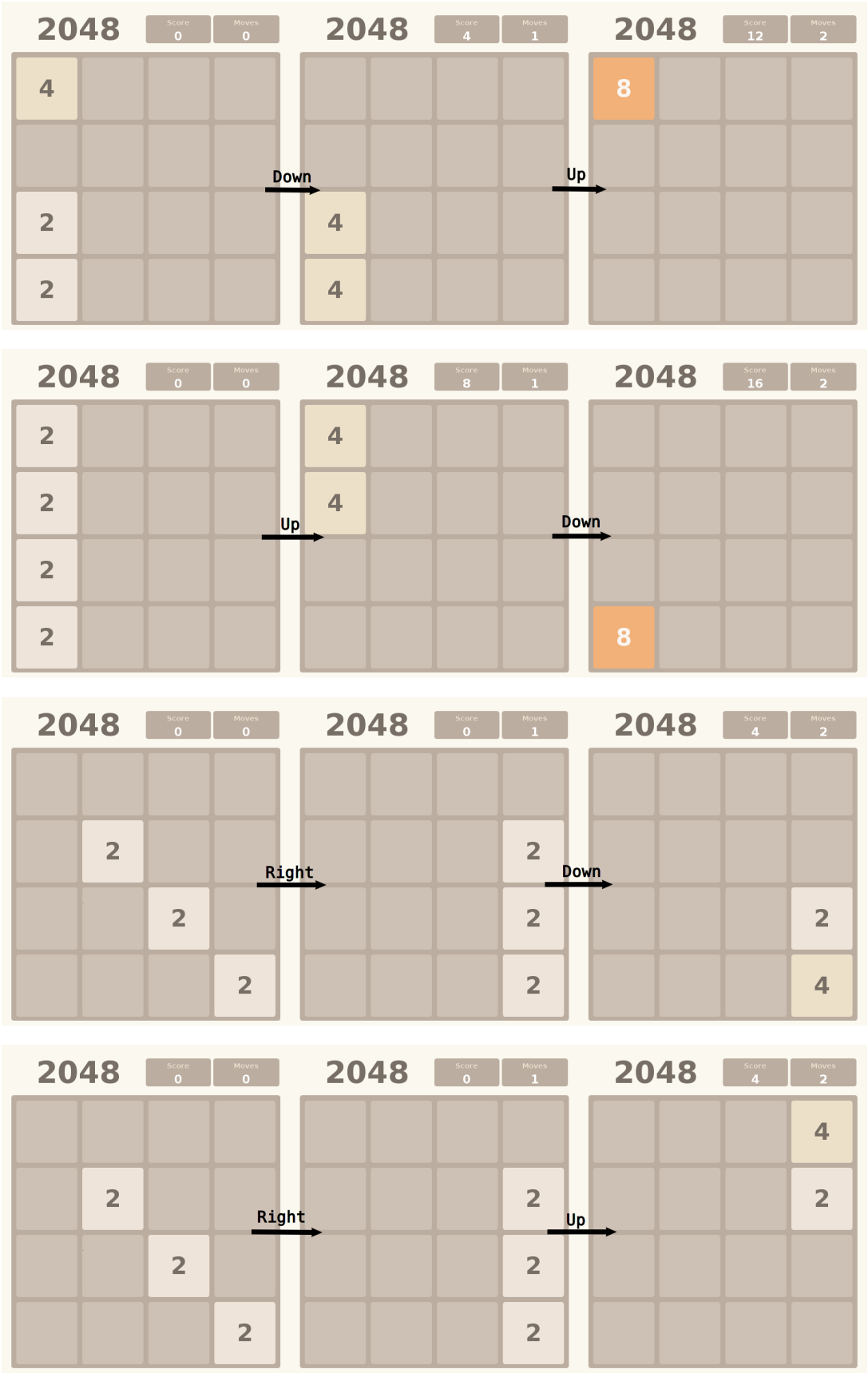

Abbildung 4: Bewegung und Verschmelzung in Beispielsituationen. Bitte beachten Sie, dass wir zur Veranschaulichung keine neuen Steine hinzugefügt haben.

3 Main

Sobald Sie den Simulator implementiert haben können Sie das Spiel auch selbst über das Userinterface spielen und ausprobieren. Führen Sie dazu die Main-Methode in der Klasse `TTFE.java` aus. Das Programm akzeptiert vier optionale Parameter in beliebiger Reihenfolge:

1. Der Seed für den Zufallszahlengenerator, bei auslassen 0: `--seed N`
2. Die Breite des Spielfelds, bei auslassen 4: `--width N`
3. Die Höhe des Spielfelds, bei auslassen 4: `--height N`
4. Ob ein Mensch oder Computer spielen soll, bei auslassen h(uman): `--player [h | c]`

Die Steuerung des Spiels funktioniert wie in der Online-Version über die Pfeiltasten.

4 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 **15.06.2021, 23:59**.

Abgabe Implementierung
: git push bis zum **22.06.2021, 23:59**.

5 Java Projekte – Visual Studio Code

Dieses und die beiden verbleibenden Projekte werden von Ihnen in Java implementiert. Wir empfehlen Ihnen die Projekte in Visual Studio Code zu bearbeiten. Die dazu erforderlichen Erweiterungen sind bereits auf der VM installiert und die zugehörigen Konfigurationsdateien werden mit dem Projekt mitgeliefert.

5.1 Kompilieren in Visual Studio Code

Die Projekte werden von Visual Studio Code automatisch neu kompiliert wenn Tests ausgeführt oder von einer Main-Methode gestartet wird. Im Gegensatz zu den C- Projekten gibt es hier also keinen Kompilierungsbefehl den Sie ausführen müssen.

5.2 Testen und Debuggen in Visual Studio Code

Das Ausführen und Debuggen einzelner Tests ist am einfachsten über das Test-Panel () möglich. Hier werden Ihnen alle im Projekt befindlichen Tests aufgelistet, gruppiert nach dem Java-Paket und der Java-Klasse in welchem sie sich befinden. Wie bereits erwähnt befinden sich ihre Tests im `ttfe.tests` Paket. Nach Ausführung eines Tests öffnet sich ein Editorfenster mit einer Zusammenfassung. Hier können Sie einsehen welche Tests bestanden wurden, sowie zusätzliche Informationen über nicht bestandene Tests einsehen die Ihnen beim Debuggen nützlich sein können. Jeder neue Test den Sie anlegen wird im Test-Panel gelistet.

Viel Erfolg!

Literatur

[1] `https://play2048.co/`

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