**Tetris Game Dart**



Dokumentation für das Modul “Webtechnologie Projekt”

Florian Jeger

Christoph Jeger

Matthias Steffen

SoSe 2017

**Inhaltsverzeichnis**

[**1.** **Einleitung** 4](#_Toc486694056)

[**2.** **Anforderungen und abgeleitetes Spielkonzept** 5](#_Toc486694057)

[2.1 Anforderungen 5](#_Toc486694058)

[2.2 Spielkonzept des Tetris Games 7](#_Toc486694059)

[**3.** **Architektur und Implementierung** 8](#_Toc486694060)

[3.1 Model 8](#_Toc486694061)

[3.1.1 TetrisGame 8](#_Toc486694062)

[3.1.2 Tetromino 10](#_Toc486694063)

[3.1.3 Level 12](#_Toc486694064)

[3.1.4 Cell 13](#_Toc486694065)

[3.2 TetrisView 14](#_Toc486694066)

[3.2.1 HTML-Dokument 14](#_Toc486694067)

[3.2.2 TetrisView als Schnittstelle zum HTML-Dokument 15](#_Toc486694068)

[**3.3** **TetrisController** 16](#_Toc486694069)

[**4.** **Level- und Parametrisierungskonzept** 17](#_Toc486694070)

[4.1 Level Konzept 17](#_Toc486694071)

[4.2 Parametrisierung Konzept 17](#_Toc486694072)

[**5.** **Nachweis der Anforderungen** 20](#_Toc486694073)

[5.1 Nachweis der funktionalen Anforderungen 20](#_Toc486694074)

[5.2 Nachweis der Dokumentationsanforderungen 20](#_Toc486694075)

[5.3 Nachweis der Einhaltung technischer Randbedingungen 21](#_Toc486694076)

[5.4 Verantwortlichkeiten im Projekt 22](#_Toc486694077)

**Abbildungsverzeichnis**

[Abbildung 1: Die Tetris-Bausteine I, J, L, O, S, T und Z 7](#_Toc486683063)

[Abbildung 2: Spielprinzip von Tetris Game 7](#_Toc486683064)

[Abbildung 3: Kollisionserkennung der Tetrominoes 12](#_Toc486683065)

**Tabellenverzeichnis**

[Tabelle 1: Anforderungen 6](#_Toc486683067)

[Tabelle 2: Punktesystem für Tetris 17](#_Toc486683068)

[Tabelle 3: Nachweis der funktionalen Anforderungen 20](#_Toc486683069)

[Tabelle 4: Nachweis der Dokumentationsanforderungen 21](#_Toc486683070)

[Tabelle 5: Nachweis der technischen Randbedingungen 22](#_Toc486683071)

[Tabelle 6: Projektverantwortlichkeiten 22](#_Toc486683072)

**Programm-Listings**

[Listing 1: HTML Basisdokument des Spiels 15](#_Toc486683073)

[Listing 2: Beispiel Parametrisierung von einem Level 18](#_Toc486683074)

[Listing 3: Parametrisierung des Spielfeldes 18](#_Toc486683075)

[Listing 4: Beispiel Parametrisierung eines Tetrominoes 19](#_Toc486683076)

# **Einleitung**

Diese Dokumentation erläutert den Aufbau des Tetris Games. Im Kapitel 2, ist das Spielkonzept und die zu erfüllenden Anforderungen zu finden. Die Umsetzung des Spielkonzepts basiert auf eine Model-View-Controller Architektur und wird im Kapitel 3 dargestellt. Das Kapitel 4 befasst sich mit dem Spiellevel und den zu definierenden Spielparametern. Abschließend in Kapitel 5 geht es um den Nachweis der Anforderungen.

# **Anforderungen und abgeleitetes Spielkonzept**

## **Anforderungen**

Das Tetris Game soll folgende in Tabelle 1 aufgeführten funktionalen Anforderungen,

Dokumentationsanforderungen und technischen Randbedingungen erfüllen.

|  |  |  |
| --- | --- | --- |
| **Id** | **Kurztitel** | **Anforderung** |
| AF-1 | Einplayer  Game | Das Spiel soll ein Einplayer Game sein (Mehrplayer Konzepte können als Einplayer Game realisiert werden, wenn Spieler durch “künstliche Intelligenzen” gesteuert werden. Beachten Sie dabei bitte, dass abhängig vom Spielkonzept die Komplexität des Spiels erheblich steigt, denken sie bspw. an Schach. Es bietet sich an, sich von alten Arcade Klassikern inspieren zu lassen.) |
| AF-2 | 2D Game | Das Spiel soll konzeptionell auf einem 2D-Raster basieren. |
| AF-3 | Levelkonzept | Das Spiel sollte ein Levelkonzept vorsehen. |
| AF-4 | Parametrisierungs-  konzept | Das Spiel sollte ein Parameterisierungskonzept für relevante Spielparameter vorsehen. |
| AF-5 | Mobile Browser | Das Spiel muss auf SmartPhone Browsern spielbar sein. |
| **Dokumentationsanforderungen** | | |
| D-1 | Dokumentations-  vorlage | Die Dokumentation soll sich an vorliegender Vorlage orientieren. |
| D-2 | Projekt-dokumentation | Das Spiel muss geeignet dokumentiert sein, so dass es von projektfremden Personen fortgeführt werden könnte. |
| D-3 | Quelltext-dokumentation | Der Quelltext des Spiels muss geeignet dokumentiert sein und mittels schriftlicher Dokumentation erschließbar und  verständlich sein. |
| D-4 | Libraries | Alle verwendeten Libraries sind aufzuführen und deren Notwendigkeit zu begründen. |
| **Technische Randbedingungen** | | |
| TF-1 | No Canvas | Die Darstellung des Spielfeldes sollte ausschließlich mittels DOM-Tree Techniken erfolgen. Die Nutzung von  Canvas-basierten Darstellungstechniken ist **explizit** untersagt. |
| TF-2 | Levelformat | Level sollten sich mittels deskriptiver Textdateien definieren lassen (z.B. mittels CSV, JSON, XML, etc.), so dass  Level-Änderungen ohne Sourcecode-Änderungen des Spiels realisierbar sind. |
| TF-3 | Parameterformat | Spielparameter sollten sich mittels deskriptiver Textdateien definieren lassen (z.B. mittels CSV, JSON, XML, etc.), so dass Parameter-Änderungen ohne Sourcecode-Änderungen des Spiels realisierbar sind. |
| TF-4 | HTML + CSS | Der View des Spiels darf ausschließlich mittels HTML und CSS realisiert werden. |
| TF-5 | Gamelogic in  Dart | Die Logik des Spiels muss mittels der Programmiersprache Dart realisiert werden. |
| TF-6 | Browser  Support | Das Spiel muss im Browser Chromium/Dartium (native Dart Engine) funktionieren. Das Spiel muss ferner in allen anderen Browserrn (JavaScript Engines) ebenfalls in der JavaScript kompilierten Form funktionieren (geprüft wird ggf. mit Safari, Chrome und Firefox). |
| TF-7 | MVC  Architektur | Das Spiel sollte einer MVC-Architektur folgen. |
| TF-8 | Erlaubte  Pakete | Erlaubt sind alle dart:\* packages, sowie das Webframework start. |
| TF-9 | Verbotene  Pakete | Verboten sind Libraries wie Polymer oder Angular. (Sollten Sie Pakete verwenden wollen, die außerhalb der erlaubten  Pakete liegen, holen Sie sich das Go ab, begründen sie bitte, wieso sie das Paket benötigen). |
| TF-10 | No Sound | Das Spiel muss keine Sounds unterstützen. |

Tabelle 1: Anforderungen

## **Spielkonzept des Tetris Games**



Abbildung 1: Die Tetris-Bausteine I, J, L, O, S, T und Z

In Abbildung 1 sind die Tetris-Bausteine zu sehen, die für das Spiel vorgesehen sind. Die Tetris-Bausteine werden zufällig generiert, d.h. es gibt eine 1/7-Wahrscheinlichkeit, das z.B. der I-Baustein vorkommt.



Abbildung 2: Spielprinzip von Tetris Game

Abbildung 2 zeigt das Spielprinzip des Tetris-Games. Der Spieler muss vom oberen Rand des rechteckigen Spielfeldes herunterfallende Tetris-Bausteine so platzieren, dass sie am unteren Rand horizontal möglichst lückenlose Reihen bilden. Dazu kann der Spieler die Tetrominoes (Tetris-Bausteine) in 90-Grad-Schritten drehen und oder sie links und rechts verschieben. Sobald eine Reihe komplett ist, verschwindet diese. Alle Reihen, die darüber liegen, rücken nach unten und geben einen Teil des Spielfeldes wieder frei. Sollte es dem Spieler gelingen gleichzeitig mehrere Reihen zu tilgen, erhält der Spieler eine höhere Punktezahl pro Reihe, als für das Tilgen einer einzelnen Reihe. Wenn eine bestimmte Anzahl entfernter Reihen erreicht worden ist, steigt der Spieler im Level auf und die Fallgeschwindigkeit der Tetrominoes wird erhöht. Das Spiel endet, sobald sich die nicht abgebauten Reihen, also jene mit Lücken, bis zum oberen Spielfeldrand aufgetürmt haben.

# **Architektur und Implementierung**

Die Architektur von Tetris Game folgt dem Model-View-Controller Prinzip. Softwaretechnisch gliedert sich die Spiellogik in mehrere Komponenten (Klassen) mit spezifischen funktionalen Verantwortlichkeiten. Eine zentrale Rolle für die Spielsteuerung hat der Controller (Klasse TetrisController).

Der Controller kann

• Nutzerinteraktionen (insbesondere Betätigen von Buttons) sowie

• Zeitsteuerung (fallen des Tetrominoes)

erkennen und in entsprechende Modelinteraktionen umsetzen.

Der Controller wird in Abschnitt 3.3, die View in Abschnitt 3.2 und das Model in Abschnitt 3.3 erläutert.

## **Model**

Aus dem Spielkonzept des Abschnitts 2.2 wurden Tetrominoes abgeleitet. Ein Spiel (Tetris Game) besteht dabei aus mindestens einem Tetromino. Das Klassendiagramm, das auch das Model beinhaltet ist im Ordner doc zu finden (Tetris Klassendiagramm.jpg).

## **TetrisGame**

Der Controller interagiert dabei nur mit dem TetrisGame. Somit greift der Controller nicht auf dahinterliegenden Entities zu. Damit ist es möglich das Spiel zu erweitern ohne den Controller ändern zu müssen.

Das TetrisGame kann dabei über folgende Attribute dem Controller Aufschluss über den aktuellen Spielzustand geben:

* \_tetromino liefert das zum Level zugehörige Tetromino.
* \_fieldHeight enthält die Feldhöhe des Spiels.
* \_fieldWidth enthält die Feldbreite des Spiels.
* \_extraFieldHeight enthält die Feldhöhe für die Extra-Feldler wie Holdbox und Nächter-Tetromino.
* \_extraFieldWidth enthält die Feldhbreite für die Extra-Feldler, wie Holdbox und Nächter-Tetromino.
* \_configReader ist ein Reader für die json Datei.
* \_levels ist eine Warteschlange für die Levels.
* \_currentLevel beschreibt das aktuelle Level.
* \_levelCount enthält die Anzahl der abgeschlossenen Level.
* \_tetrominoCount zählt die Anzahl der bereits gefallenen Tetrominoes.

* \_numberOfRowsCleared zählt Anzahl an gelöschte Reihen.
* \_tetrominoQueue ist eine Warteschlange für die Tetrominoes.

* \_tetrominoOnHold ist die Holdbox (leer oder mit Tetromino gefuellt).
* \_usedHoldBox ist dazu da, ob die Holdbox benutzt wurde oder nicht und ist am Anfang auf false gesetzt d.h. ein Tetromino wurde während eines fallenden Tetromino noch nicht getauscht. Ist usedHoldBox true dann wurde ein Tetromino während des Fallens getauscht.
* \_endlessMode checkt ob der Endlos Modus erreicht ist.
* \_field ist die interne Representation des Spielfelds.
* \_gamestate gibt an ob das Spiel läuft oder gerade stoppt.
* \_score gibt den aktuellen Punktestand an.

Ein TetrisGame Objekt

* kann mittels des Konstruktors erzeugt werden. Die Höhe und Breite werden aus der json Datei geladen.
* mit den Methoden start(), pause(), resume(), stop() wird das Spiel gestartet, pausiert, fortgesetzt oder gestoppt.
* die Methode fillTetrominoeQueue() füllt die Tetromino in die Warteschlange
* hardDropCurrentTetromino() ist für den direkten Fall des Tetrominoes zuständig.
* holdCurrentTetrominoe() legt den aktuellen Tetrominoe in die Hold Box. Falls bereits ein Tetrominoe in der Hold Box ist, wird dieser aus der Box genommen und fällt herunter.
* die Methode updateField() ist für die Aktualisierung der Interne Repräsentation des Spielfelds zuständig. Muss nach jeder Änderung am Zustand des Spielfeldes (z.B. Bewegen eines Tetrominoes) aufgerufen werden.
* mittels der Methode moveTetromino() kann der Tetromino nach links, rechts und nach unten bewegt werden. Bewegungen sind nur im Status running möglich.
* die Methode pauseTetromino() pausiert das Spiel und gibt es beim nächsten Aufruf wieder frei.
* getIndexOfCompletedRows() gibt eine Liste mit den Indizes aller Zeilen die vollständig sind zurück. Eine Reihe gilt als vollständig, falls alle ihre Zellen gefüllt und inaktiv sind.
* mittels der Methode removeCompletedRows() kann jede vollständige Reihe aus dem Spielfeld entfernt werden.
* removeRows() entfernt eine vollständige Tetromino Reihe.
* calculateScoreOfMove() berechnet wie viele Punkte das letzte vervollständigen wert ist.
* mit der Methode addLevel() können Level hinzugefügt werden.
* incrementTetrominoCount() erhöht den Tetromino Zähler.
* \_startNextLevel() startet den nächsten Level.

## **Tetromino**

Der Zustand eines Tetromino besteht aus

* \_stones Liste der einzelnen Tetromino Steine.
* \_preview Liste der einzelnen Tetromino Steine für die Vorschau des nächsten Tetromino.
* \_initialPosition Liste der Position wo der Tetromino oben erscheinen soll.
* \_model ist das Tetris Model.
* \_color Farbe des Tetrominos.
* \_dc gibt die Bewegungsrichtung in der Spalte an.
* \_dr gibt die Bewegungsrichtung in der Reihe an.
* \_state den Aktuellen Zustand.
* \_numberOfStates Anzahl der Zustände.
* \_transitions enthält die transitions (Übergänge) zu den neuen Positionen, wenn ein Tetromino gedreht wird.

Die Tetromino Klasse hat folgende Methoden:

* Tetromino() Konstruktor der Tetromino Klasse
* \_calculateInitialPosition() berechnet für den fallenden Tetromino die mittige Start Position.
* addToField() werden Tetromino Steine dem Feld hinzufügt.
* removeFromField() entfernt Tetromino Steine vom Feld.
* resetPosition() setzt den Tetromino auf seine Anfangsposition (idR. am oberen Rand des Spielfelds) zurück. Dabei wird der Rotationszustand auf den initialen Zustand zurückgesetzt.
* rotate() ist für die Rotation des Tetromino zuständig. Abstrakte Definition für die Drehung eines Tetrominos. Die Rotation ist für jeden Tetromino anders. Die richtige Drehmatrix wird in Abhängigkeit des Zustandes und der Drehrichtung ausgewählt.
* move() bewegt den Tetromino und prüft auf Kollisionen.
* \_handleCollision() handelt die Kollisionen ab.
* \_moveToNewPosition() bewegt den Tetromino zur nächsten Position.
* \_collisionWithBorder() prüft auf Kollisionen mit dem Rand.
* \_collisionWithGround() prüft auf Kollisionen mit dem Grund.
* \_collisionWithTop() prüft auf Kollisionen mit den oberen Rand.
* \_collisionWithOtherTetromino() prüft auf Kollisionen mit Tetrominoes.
* stop(), down(), left(), right() teilt dem Tetromino die Bewegung nach links, rechts, unten mit oder ob keine Bewegung stattfinden soll.

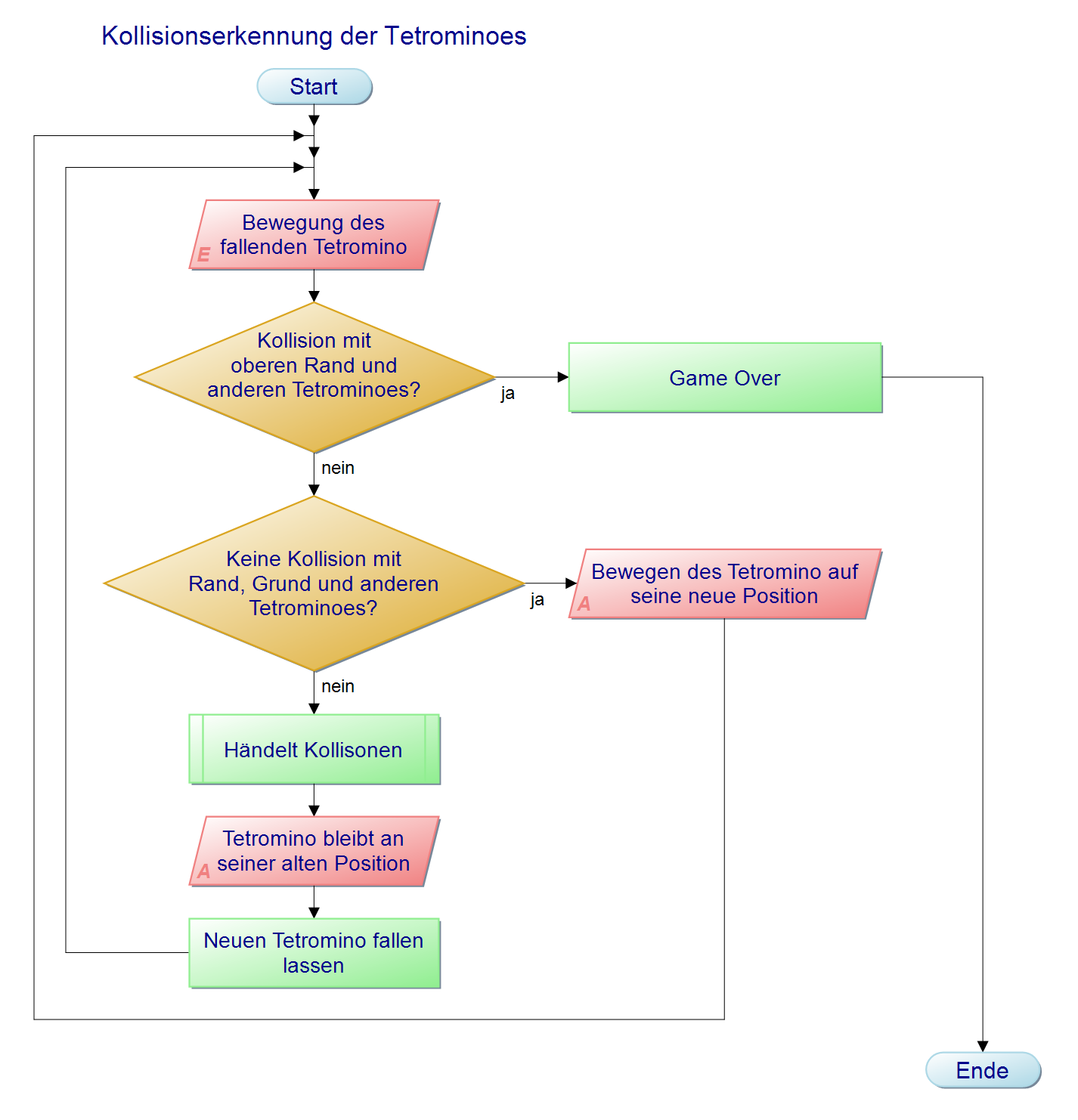


Abbildung 3: Kollisionserkennung der Tetrominoes

Die Kollisionserkennung der Tetrominoes ist in Abbildung 3 veranschaulicht dargestellt. Es werden mehrere Kollisionsarten geprüft und dann wird entweder Game Over ausgelöst, der Tetromino bleibt an seiner Position und es wird ein neuer Tetromino fallengelassen oder der Tetromino bewegt sich auf eine neue Position.

## **Level**

Ein Level enthält verschiedene Konfigurationen für das Spiel (TetrisGame). Ein Level besitzt ein Ziel, wenn dieses erreicht wurde ist das Level beendet. Der Zustand eines Levels besteht aus

* \_model das zum Level zugehörige
* \_idsOfAvailableTetrominoes enthält die eindeutigen Ids aller Tetrominoes, die in diesem Level verfügbar sind.
* \_scoreMultiplier verändert die Berechnung der Punkte die der User in diesem Level für das vervollständigen von Reihen erhält.
* \_tetrominoSpeedInMs ändert die Fallgeschwindigkeit des Tetrominos solange dieses Level aktiv ist.
* \_goal Alle Ziele für dieses Level. Jedes einzelne Ziel darf maximal einmal vorkommen.
* \_bonusPoints der Spieler erhält viele Extrapunkte, nachdem er das Ziel dieses Levels erfüllt hat.
* \_priority die Priorität bestimmt die Reihenfolge der verschiedenen Level. Das Level mit der höchsten Priorität wird als erstes gestartet.
* \_goalMetrics diese Map speichert den aktuellen Zustand aller Werte, die für das Erfüllen eines Ziels von Bedeutung sind. Sobald sich einer dieser Werte ändert, muss die Map aktualisiert werden.

Die Level Klasse hat folgende Methoden:

* Level() erstellt eine neue Level Instanz, welche nicht Konfiguriert ist. Es sollte der entsprechende LevelBuilder genutzt werden, oder das Level wird über die setter Methoden konfiguritert.
* isComplete() prüft ob das Ziel dieses Levels erfüllt wurde.
* \_initGoalMetrics() initialisiert eine neue Map mit allen Metriken die für den Fortschritt des Levels relevant sind.

## **Cell**

Definiert die interne Repräsentation des Spielfelds. Der Zustand eines Cells besteht aus

* \_isActive der fallende Tetromino der gerade aktiv ist und sich bewegen kann
* \_row Reihe
* \_col Spalte
* \_color Farbe des Tetrominoes

Die Cell Klasse hat einen Konstruktor Cell()um ein Cell Objekte zu erzeugen.

## **TetrisView**

Die TetrisView ist für die Darstellung des Spiels zuständig. Im Kern besteht die TetrisView aus einem HTML-Dokument (siehe Abschnitt 3.2.1) und einer clientseitigen Logik, die den DOM-Tree des HTML-Dokuments manipuliert (siehe Abschnitt 3.2.2). Das Klassendiagramm, das auch die View beinhaltet ist im Ordner doc zu finden (Tetris Klassendiagramm.jpg).

### **HTML-Dokument**

Die TetrisView wird im Browser durch folgendes HTML-Dokument erzeugt. Der DOM-Tree dieses HTML-Dokuments wird im Verlaufe des Spiels durch die Klasse TetrisView manipuliert um den Spielzustand darzustellen und Nutzerinteraktionen zu ermöglichen. Die Klasse TetrisView wird dabei durch das Script tetrisclient.dart als clientseitige Logik geladen.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88 | <!DOCTYPE html> <**html** lang=**"de"**> <**head**>  <**meta** charset=**"utf-8"**>  <**meta** name=**"viewport"** content=**"user-scalable=no, width=device-width, initial-scale=1.0"**>  <**title**>Tetris</**title**>  <**link** href=**"favicon.ico"** rel=**"shortcut icon"** type=**"images/x-icon"**>  <**link** rel=**"stylesheet"** type=**"text/css"** href=**"styles.css"**> </**head**> <**body**> *<!-- Container für die Startseite -->* <**div** class=**"container\_start"**>  <**div** id=**"title"**></**div**>  <**p** id=**"start"**>START</**p**>  <**p** id=**"copy"**>**&copy;** Florian Jeger, Christoph Jeger **&amp;** Matthias Steffen</**p**> </**div**>  <**div** id=**"overlay"**> </**div**>  *<!-- Container für die Nachrichten -->* <**div** class=**"container\_message"**>  <**img** id=**"logo"** src=**'img/tetris\_menu\_logo.png'**>  <**img** id=**"portrait"** src=**'img/portrait.png'**>  <**p** id=**"message"**></**p**>  <**p** id=**"continue"**>FORTSETZEN</**p**>  <**p** id=**"newGame"**>NEUES SPIEL</**p**> </**div**>  <**div** class=**"container\_game"**>  *<!-- Container für die Level Ziele -->* <**div** class=**"container\_goal"**>  <**b**>Ziel:</**b**> <**span** id=**"goalDescription"**></**span**> <**span** id=**"goalProgress"**>0</**span**>/<**span** id=**"goal"**>0</**span**>  <**br**/>  <**b**>Belonung: </**b**> <**span** id=**"bonusPoints"**>0</**span**> Punkte  </**div**>   *<!-- Tabelle für das Spielfeld -->* <**table** id=**"field"**>  </**table**>   <**div** class=**"container\_sidebox"**>  *<!-- Container für die Seitenbox -->* <**div** class=**"container\_nextstone"**>  <**h3**>Nächster Stein</**h3**>  *<!-- Tabelle für den nächsten Stein -->* <**table** id=**"nextstone"**>  </**table**>  </**div**>   <**div** class=**"container\_holdstone"**>  <**h3**>Stein halten</**h3**>  *<!-- Tabelle für den nächsten Stein -->* <**table** id=**"holdstone"**>  </**table**>  </**div**>   *<!-- Container für die Spielinformationen -->* <**div** class=**"container\_info"**>  <**h3**>Level</**h3**>  <**p** id=**"level"**>0</**p**>  <**h3**>Punkte</**h3**>  <**p** id=**"points"**>0</**p**>  </**div**>  </**div**>    *<!-- Container für die Steuerung -->* <**div** class=**"container\_control"**>  <**div** id=**"menu"**><**img** src=**'img/menu.png'**></**div**>  <**div** id=**"hard\_drop"**><**img** src=**'img/hard\_drop.png'**></**div**>  <**div** id=**"left\_rotation"**><**img** src=**'img/left\_rotation.png'**></**div**>  <**div** id=**"right\_rotation"**><**img** src=**'img/right\_rotation.png'**></**div**>  <**div** id=**"hold"**><**img** src=**'img/hold.png'**></**div**>  <**div** id=**"left"**><**img** src=**'img/left.png'**></**div**>  <**div** id=**"down"**><**img** src=**'img/down.png'**></**div**>  <**div** id=**"right"**><**img** src=**'img/right.png'**></**div**>  </**div**>   <**script** type=**"application/dart"** src=**"tetrisclient.dart"**></**script**>  <**script** data-pub-inline src=**"packages/browser/dart.js"**></**script**> </**div**> </**body**> </**html**> |

Listing 1: HTML Basisdokument des Spiels

Um das Spiel einzublenden wird diese HTML-Dokument genutzt.

### **TetrisView als Schnittstelle zum HTML-Dokument**

Alle notwendigen CSS-Gestaltungen werden in der style.css vorgenommen. Im tetrisclient.dart Script wird die Applikationslogik geladen. Für Browser die nicht Dart-fähig sind, wird gemäß den Dart-Konventionen die dart.js geladen, damit wird die Dart Logic in Javascript Engines zu Ausführung gebracht.

## **TetrisController**

Für die Ablaufsteuerung des Spiels ist der TetrisController zuständig. Das Klassendiagramm, das auch den TetrisController beinhaltet ist im Ordner doc zu finden (Tetris Klassendiagramm.jpg).

Der TetrisController hat dabei folgende Attribute:

* tetrominoSpeed bestimmt die Geschwindigkeit des Tetrominoes.
* game das zu diesem Controller zugehörige Model.
* \_view die zu diesem Controller zugehörige Ansicht.
* tetrominoTrigger ist ein periodischer Timer für das Spiel.
* \_configReader ist ein Reader welcher die Konfiguration für die zu diesem Controller zugehörige Spielinstanz bereitstellt.
* \_currentLevel zählt die Level.

Die TetrisController Klasse hat folgende Methoden:

* \_moveTetromino() bewegt den Tetromino.
* \_registerControlCallbacks registriert die Callbacks für die Steuerung des Spiels.
* \_increaseTetrominoSpeed() erhöht die Geschwindigkeit für die fallenden Tetrominoes.
* \_newGame() initialisiert ein neues Spiel.
* TetrisController() erstellt einen neuen TetrisContoroller, dessen Model nach der Konfiguration des übergebenen Reader konfiguriert wird.

# **Level- und Parametrisierungskonzept**

## **Level Konzept**

Die folgende Tabelle zeigt wie unsere Punkte berechnet werden:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Level** | **Punkte für 1 Reihe** | **Punkte für 2 Reihen** | **Punkte für 3 Reihen** | **Punkte für 4 Reihen** |
| 1 | 40 | 100 | 300 | 1200 |
| 2 | 80 | 200 | 600 | 2400 |
| 3 | 120 | 300 | 900 | 3600 |
| … | … | … | … | … |
| 10 | 400 | 900 | 3000 | 12000 |
| n | 40\*(n+1) | 100\*(n+1) | 300\*(n+1) | 1200\*(n+1) |

Tabelle 2: Punktesystem für Tetris

Des Weiteren bekommt der Spieler pro erreichtes Level Bonuspunkte. Der Spieler bekommt beim Abschluss von Level 1 1000 Bonuspunkte und für jedes weitere Level werden 250 Punkte aufaddiert. Somit bekommt der Spieler beim Abschluss von Level 2 1250 Punkte und für Level 3 1500 extra Punkte und so weiter.

Um ein Level aufzusteigen muss der Spieler bestimmte Ziele erfüllen. Hierbei gibt es zwei verschiedene Zielen die erfüllt werden müssen. Hierbei handelt es sich um eine bestimmte Anzahl von Reihen tilgen oder es muss eine bestimmte Anzahl an Tetrominoes gespielt werden.

Die Bonuspunkte und die Ziele zum Erreichen eines Levels werden im Spiel angezeigt.

Die Fallgeschwindigkeit erhöht sich bei jedem Level, damit das Spiel schwerer wird. Die Start Fallgeschwindigkeit liegt bei 1000ms, d.h. jede Sekunde fallen die Tetrominoes um eine Einheit. Jedes Level reduziert sich die Fallgeschwindigkeit um 200ms und die Tetrominoes fallen dementsprechend schneller.

## **Parametrisierung** **Konzept**

Alle Parameter lassen sich in der game-config.json(im Ordner web) Datei anpassen bzw. modifizieren.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | {  "id": "level2",  "availibleTetrominoes": [  "ITetromino",  "TTetromino",  "JTetromino",  "LTetromino",  "OTetromino",  "STetromino",  "ZTetromino"  ],  "scoreMultiplier": 2.0,  "tetrominoSpeedInMs": 800,  "goal": {  "numberOfRowsCleared": 4.0  },  "bounsPoints": 1250,  "priority": 80  }, |

Listing 2: Beispiel Parametrisierung von einem Level

Im Listing 2 wird anhand des Level 2 beispielhaft gezeigt wie ein Level parametrisiert wird. Die id gibt an um welches Level es sich handelt. Die Tetrominoes die in dem Level verfügbar sein sollen werden unter availibleTetrominoes angegeben. Der scoreMultiplier gibt an um wie viel die Punkte multipliziert werden soll. Die Fallgeschwindigkeit wird unter tetrominoSpeedInMs in Millisekunden angegeben. Unter goal werden die Ziele des Levels angegeben, in diesem Beispiel muss der Spieler 4 Reihen tilgen um das nächste Level zu erreichen, siehe Zeile 15. Unter bonusPoints werden die Bonuspunkte für das erfolgreiche beenden des Levels festgelegt. Die priority gibt an welche Priorität das Level hat, damit zum Beispiel Level 2 nicht vor Level 1 kommt.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {  "gameConfiguration": {  "id":"modelDefault",  "fieldWidth": 10,  "fieldHeight": 18  }, |

Listing 3: Parametrisierung des Spielfeldes

Unter Listing 3 wird die Parametrisierung des Spielfeldes angeben. Dazu muss unter fieldWidth und fieldHeight die Breite und Höhe des Spielfeldes angeben werden.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | "tetrominoes": [  {  "id": "ITetromino",  "stones":[  { "row" : 0, "col" : -2 },  { "row" : 0, "col" : -1 },  { "row" : 0, "col" : 0},  { "row" : 0, "col" : 1}  ],  "preview":[  { "row" : 2, "col" : 0},  { "row" : 2, "col" : 1},  { "row" : 2, "col" : 2},  { "row" : 2, "col" : 3}  ],  "transitions":[  [[-3, 1], [-2, 0], [-1, -1], [0, -2]],  [[3, -1], [2, 0], [1, 1], [0, 2]],  [[-3, 1], [-2, 0], [-1, -1], [0, -2]],  [[3, -1], [2, 0], [1, 1], [0, 2]]  ],  "powerUps":[],  "color": "cyan"  }, |

Listing 4: Beispiel Parametrisierung eines Tetrominoes

Um ein Tetromino zu erstellen muss eine Parametrisierung vorgenommen werden, siehe Listing 4. Unter id wird der Name des Tetromino angegeben, die dazu gebraucht wird, um bei der Level Parametrisierung anzugeben, ob der Tetromino in dem Level vorkommen soll oder nicht. Jeder Stein muss extra angegeben werden, d.h. aus mehreren Steinen ergibt sich dann ein Tetromino und muss unter stones und preview(für das Vorschaufenster) angegeben werden. Für die Drehung des Tetromino muss eine Dreh Matrix angeben werden unter transitions. Bei powerUps wird angeben, ob der Stein ein Power Up hat oder nicht. Die Farbe des Tetromino muss unter color angegeben werden.

# **Nachweis der Anforderungen**

Nachfolgend wird erklärt wie die im Kapitel 2 aufgeführten funktionalen Anforderungen eingehalten bzw. erfüllt werden. Dies wird nachfolgend argumentativ erfolgen und wird nicht durch Testfall-getriebene Nachweisführung erfolgen. Abschließend wird angegeben, wer im Team welche Verantwortlichkeiten hatte.

## **Nachweis der funktionalen Anforderungen**

Nachfolgend erfolgt der Nachweis der Einhaltung funktionaler Anforderungen.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **Kurztitel** | **Erfüllt** | **Teilw. erfüllt** | **Nicht erfüllt** | **Erläuterung** |
| AF-1 | Einplayer Game | x |  |  | Das Tetris Game ist ein Einpersonen Spiel, wie aus dem in Abschnitt 2.2 dargestellten Spielkonzepts hervorgeht. |
| AF-2 | 2D Game | x |  |  | Das Tetris Game wird auf einem 2D-Raster gespielt, wie aus dem in Abschnitt 2.2 dargestellten Spielkonzepts hervorgeht. |
| AF-3 | Levelkonzept |  |  |  |  |
| AF-4 | Parametrisierungs-  konzept |  |  |  |  |
| AF-5 | Mobile Browser | x |  |  | Das Spiel ist in Mobile Browsern spielbar. Es wurde in Chrome, Safari und Firefox erfolgreich zur Ausführung gebracht. |

Tabelle 3: Nachweis der funktionalen Anforderungen

## **Nachweis der Dokumentationsanforderungen**

Nachfolgend erfolgt der Nachweis der Einhaltung der Dokumentationsanforderungen.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **Kurztitel** | **Erfüllt** | **Teilw. erfüllt** | **Nicht erfüllt** | **Erläuterung** |
| D-1 | Dokumentations-vorlage | x |  |  | Vorliegende Dokumentation diente als Vorlage für Spieldokumentationen. |
| D-2 | Projektdokumentation | x |  |  | Vorliegende Dokumentation erläutert die übergeordneten Prinzipien und verweist an geeigneten Stellen auf die Quelltextdokumentation. |
| D-3 | Quelltext-dokumentation | x |  |  | Es wurden alle Methoden und Datenfelder, Konstanten durch Inline-Kommentare erläutert. |
| D-4 | Libraries | x |  |  | Alle genutzten Libraries werden in der pubspec.yaml der Implementierung aufgeführt. Da nur die zugelassenen Pakete genutzt wurden, sind darüber hinaus keine weiteren Erläuterungen, warum welche Pakete genutzt wurden, erforderlich. |

Tabelle 4: Nachweis der Dokumentationsanforderungen

## **Nachweis der Einhaltung technischer Randbedingungen**

Nachfolgend erfolgt der Nachweis der Einhaltung der vorgegebenen technischen Randbedingungen.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **Kurztitel** | **Erfüllt** | **Teilw. erfüllt** | **Nicht erfüllt** | **Erläuterung** |
| TF-1 | No Canvas | x |  |  | Die Klasse TetrisView nutzt keinerlei Canvas basierten DOM-Elemente |
| TF-2 | Levelformat |  |  |  |  |
| TF-3 | Parameterformat |  |  |  |  |
| TF-4 | HTML + CSS | x |  |  | Die View des Spiels beruht ausschließlich auf HTML und CSS |
| TF-5 | Gamelogic in Dart | x |  |  | Die Logik des Spiels ist in der Programmiersprache Dart realisiert worden. |
| TF-6 | Browser Support | x |  |  | Das Spiel funktioniert im Browser Chromium/Dartium. Das Spiel muss ferner in allen anderen Browsern funktionieren. Geprüft wurden Safari, Chrome und Firefox. |
| TF-7 | MVC Architektur | x |  |  | Das Spiel folgt durch Ableitung mehrerer Modell-Klassen, einer View Klasse und dem zentralen Controller einer MVC-Architektur. Der Controller triggert das Modell und die View. Die View greift zudem auf das Model nur lesend und nicht manipulierend zu. |
| TF-8 | Erlaubte Pakete | x |  |  | Es sind nur dart:\* packages, sowie das Webframework start genutzt worden. Siehe pubspec.yaml der Implementierung. |
| TF-9 | Verbotene Pakete | x |  |  | Es sind keine Pakete, außer den erlaubten genutzt worden. Siehe pubspec.yaml der  Implementierung. |
| TF-10 | No Sound | x |  |  | Das Spiel hat keine Soundeffekte. |

Tabelle 5: Nachweis der technischen Randbedingungen

## **Verantwortlichkeiten im Projekt**

Nachfolgend erfolgt eine Übersicht über die Aufgabenteilung.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Komponente** | **Detail** | **Asset** | **Florian Jeger** | **Christoph Jeger** | **Matthias Steffen** | **Anmerkungen** |
| Modell | Spielfeld | /lib/src/model/  Cell.dart |  |  |  |  |
|  | Tetris Spiel | /lib/src/model/  TetrisGame.dart |  |  |  |  |
|  | Tetromino | /lib/src/model/  Tetromino.dart |  |  |  |  |
| View | HTML-Dokument | web/index.html |  |  |  |  |
|  | Gestaltung | web/styles.css |  |  |  |  |
|  | Bilder | web/img/\* |  |  |  |  |
|  | Viewlogik | /lib/src/view/  TetrisView.dart |  |  |  |  |
| Controller | Eventhandling | lib/src/controller/  TetrisController.dart |  |  |  |  |
|  | Parametrisierung |  |  |  |  |  |
|  | Level |  |  |  |  |  |
| Dokumentation | Tetris- Documentation | doc/\*.\* |  |  |  |  |

Tabelle 6: Projektverantwortlichkeiten

V = verantwortlich (hauptdurchführend, kann nur einmal pro Zeile vergeben werden)

U = unterstützend (Übernahme von Teilaufgaben)