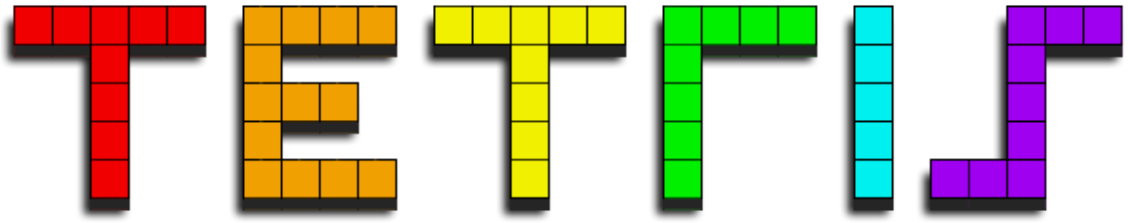
Tetris Game Dart



Dokumentation für das Modul “Webtechnologie Projekt”

Florian Jeger

Christoph Jeger

Matthias Steffen

SoSe 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programm-Listings

Quellenverzeichnis

Abbildung 1: Die Tetris-Bausteine I, J, L, O, S, T und Z, Wikipedia, https://de.wikipedia.org/wiki/Datei:Tetrominoes_IJLO_STZ_Worlds.svg), 04.07.2017

Abbildung 1: Spielprinzip von Tetris Game, Wikipedia, https://de.wikipedia.org/wiki/Datei:Tetris-gravity-simple.svg), 04.07.2017

1. 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.

2. Anforderungen und abgeleitetes Spielkonzept

2.1 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

2.2 Spielkonzept des Tetris Games

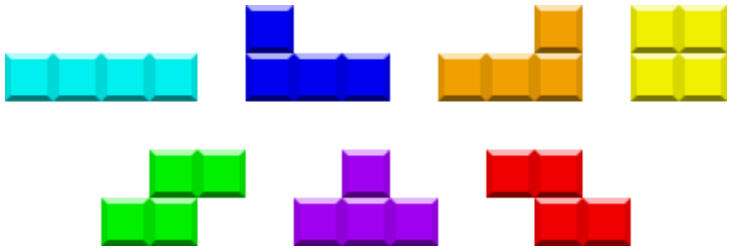


Abbildung 2: Die Tetris-Bausteine I, J, L, O, S, T und Z

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, dass z.B. der I-Baustein vorkommt.

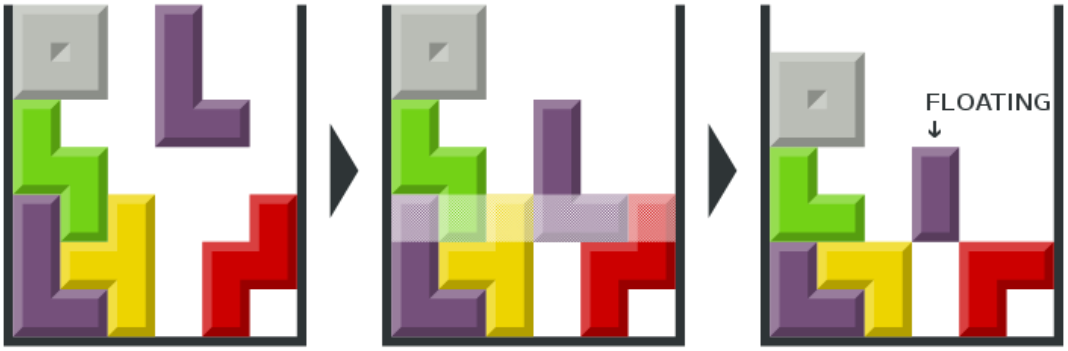


Abbildung 3: Spielprinzip von Tetris Game

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, wird diese getilgt. 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 Reihen oder eine bestimmte Anzahl von Tetromioes gespielt worden ist, steigt der Spieler im Level auf und die Fallgeschwindigkeit der Tetrominoes wird erhöht. Das Spiel endet, sobald sich die nicht getilgten Reihen, also jene mit Lücken, bis zum oberen Spielfeldrand aufgetürmt haben.

3. Architektur und Implementierung

Die grundlegende Architektur des Spiels beruht auf dem Model-View-Controller (MVC) Pattern. Durch die Verwendung dieses Design Patterns entsteht der Vorteil, dass die unterschiedlichen Aufgaben klar getrennt und der Code so gut strukturiert werden kann. Das Modell ist für die Implementierung der Spiellogik und das Verwalten des Spielstandes verantwortlich. Die dazu notwendigen Entitäten und Funktionen werden nach außen hin gekapselt und sind nur über öffentliche Schnittstellen nutzbar. Dies bietet den Vorteil, dass die Implementierung des Modells geändert werden kann ohne das restliche Design zu verändern, solange die öffentlichen Schnittstellen des Modells sich nicht ändern. Die View ist für die visuelle Darstellung des Modellzustandes zuständig. Es wird ein HTML Dokument verwendet, welches vom Browser gerendert wird. Das HTML Dokument wird durch die Klasse TetrisView gekapselt, somit ist das Design nicht von der Darstellung in Form eines HTML Dokuments abhängig. Der Controller nimmt Benutzereingaben entgegen und manipuliert das Modell über die Schnittstellen entsprechend der Nutzerinteraktion. Des Weiteren ist der Controller für das periodische Bewegen des aktuell Tetrominoes verantwortlich.

Zusätzlich zum MVC Pattern wird das Builder Pattern zu Instanziierung komplexer Entitäten eingesetzt. Dieses Pattern bietet den Vorteil, dass die Konstruktoren von komplexen Klassen trotz einer großen Anzahl von Parametern übersichtlich bleiben. Außerdem kann so die Instanziierung von Klassen gekapselt werden, was im restlichen Code die Lesbarkeit erhöht und die Konfiguration von Instanzen flexibler macht.

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.

3.1 Model

Aus dem Spielkonzept des Abschnitts 2.2 lassen sich mehrere Entitäten herleiten, welche für die Spiellogik unerlässlich sind. Diese Entitäten gehören konzeptionell zum Modell sind aber in eigene Klassen gekapselt. Das Klassendiagramm, das auch das Model beinhaltet ist im Ordner doc zu finden (Tetris Klassendiagramm.jpg).

3.1.1 TetrisGame

Die Klasse TetrisGame implementiert die Hauptlogik des Tetris Spiels und stellt öffentliche Schnittstellen zur Manipulation des Spielzustandes bereit. Zur Verwaltung des Spielfeldes wird eine zweidimensionale Liste von Instanzen der Klasse Cell genutzt.

Die Klasse Cell ist eine einfache Datenstruktur die wichtigen Informationen über den Zustand eines Blocks des Spielfeldes hält. Aus ihr kann entnommen werden, ob sich in der entsprechenden Zelle ein Tetromino befindet oder ob die Zelle leer ist. Falls sich in der Zelle Teile eines Tetrominoes befinden, kann anhand des isActive Feldes festgestellt werden, ob der sich in dieser Zelle befindliche der zu diesem Zeitpunkt herunterfallende Tetromino ist.

Die Klasse TetrisGame ist außerdem für die Verwaltung der Tetrominoes zuständig. Tetrominoes sind in einer eigenen Klasse gekapselt. Zur Verwaltung der Tetrominoes nutzt das TetrisGame eine Queue. Diese wird mit allen im momentanen Level verfügbaren Tetrominoes in einer zufälligen Reihenfolge befüllt. Dann wird für jeden Tetromino der herunterfallen soll die pop Operation der Queue ausgeführt und dieser Tetromino wird in dem Feld currentTetromino gespeichert. Die Verwaltung der verschiedenen Level geschieht analog. Es wird jedoch eine Priority Queue genutzt, um die vom Nutzer gewünschte Reihenfolge der Level zu garantieren.

Die Klasse TetrisGame stellt die Schnittstellen für die Bewegung und Halten des aktuellen Tetromino nach außen hin bereit. Des Weiteren bietet sie Schnittstellen, um das zu pausieren und nach einer Pause fortzusetzen.

Der Controller interagiert dabei nur mit dem TetrisGame. Somit greift der Controller nicht auf dahinterliegenden Klassen 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.
- removeTetrominoFromCells() entfert Tetromino Bestandteile an alle in cells übergebenen Positionen

3.1.2 Tetromino

Die Tetromino Klasse kapselt alle Felder und Funktionen, die einen Tetromino definieren. Die wichtigsten Felder sind dabei:

- _stones eine Liste von Objekten, welche die aktuelle Position aller Bestandteile dieses Tetrominoes speichern.
- _preview eine Liste von Objekt die beschreibt, wie dieser Tetromino in der Vorschau Box angezeigt wird.
- _initialPosition Liste der Position wo der Tetromino oben erscheinen soll.
- _model ist das Tetris Model.
- _color Farbe des Tetrominos.
- dc/dr die Komponenten eines zweidimensionalen Vektors, welche die Bewegungsrichtung des Tetrominoes beschreibt.
- _state speichert in welchem Rotationszustand sich der Tetromino befindet.
- _numberOfStates Anzahl der Zustände.
- _transitions eine Liste von Drehmatrizen, die das Rotationsverhalten des Tetrominoes beschreiben.

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.

Die beiden wichtigsten Probleme, welche die Tetromino Klasse löst sind die Rotation von Tetrominoes und die Kollisionserkennung.

Um das Problem der Rotation von Tetrominoes möglichst generisch zu lösen, wurde ein Algorithmus basierend auf Rotationsmatrizen entwickelt. Ein Tetromino besitzt 4 unterschiedliche Zustände, diese basieren auf dem Winkel um den der Tetromino rotiert (0°, 90°, 180°, 270°). Für die Transition von einem Zustand in den nächsten gibt es jeweils eine Rotationsmatrix, die für jeden Stein des Tetrominoes den Unterschied der neuen Position in Abhängigkeit der vorherigen Position beschreibt. Abhängig von der Rotationsrichtung, wird der in der Rotationsmatrix beschriebene Unterschied zu den einzelnen Steinen des Tetrominoes addiert oder subtrahiert.

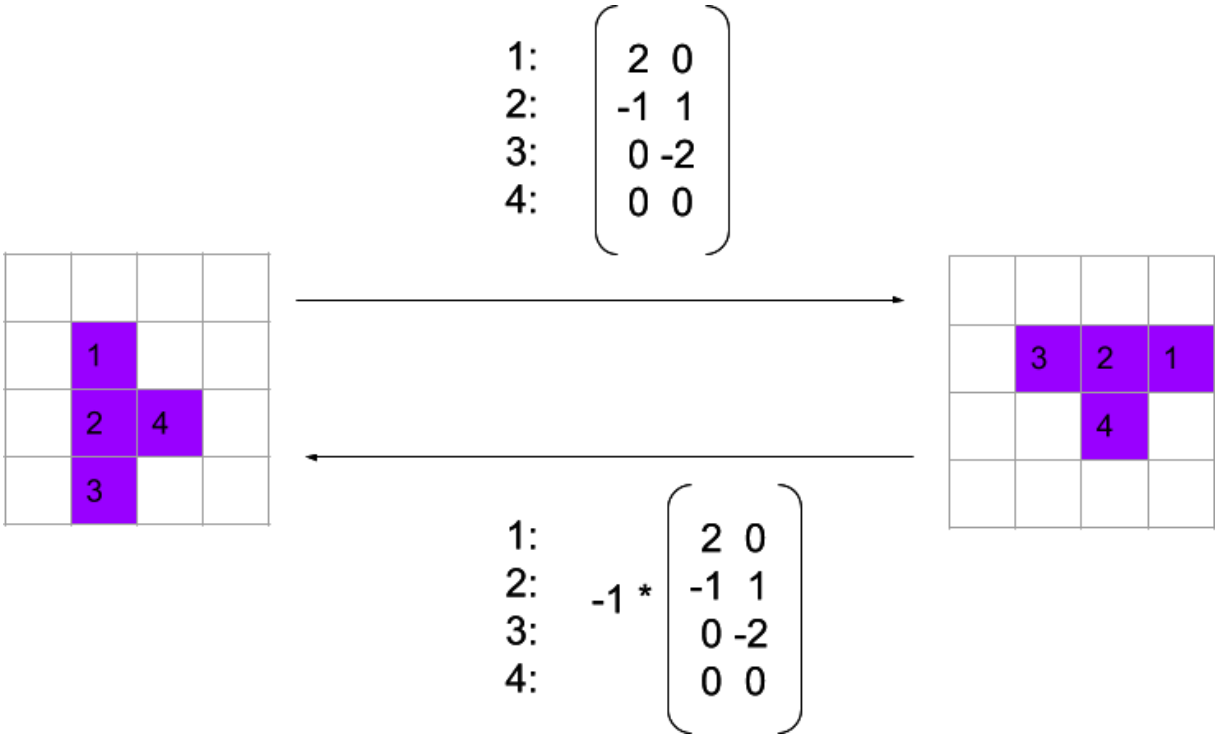


Abbildung 4: Beispielhafte Drehung eines Tetrominoes

In Abbildung 3 ist beispielhaft die Drehung eines Tetrominoes gezeigt. Dazu sind die Steine, aus denen der Tetromino besteht durchnummeriert (eins bis vier). Die Dimension der Drehmatrix hängt nur von der Anzahl der Steine (n) ab und ist n x 2. Um die Position eines Steins nach der Drehung zu berechnen wird dessen Position vor der Drehung mit der entsprechenden Zeile der Matrix addiert. So wird für eine Drehung im Uhrzeigersinn z.B. der zweite Stein um eine Zeile nach oben und eine Spalte nach rechts bewegt.

Der Zustand in dem sich ein Tetromino befindet wird gespeichert, damit bei der nächsten Rotation die Richtige Drehmatrix ausgewählt werden kann. Wird ein Tetromino z.B. von Zustand drei nach Zustand zwei überführt, so muss die Matrix M_2 verwendet werden und da es sich um eine Drehung gegen den Uhrzeigersinn handelt muss die Matrix noch mit -1 multipliziert werden. Analog können alle Zustandswechsel zwischen zwei benachbarten Zuständen berechnet werden.

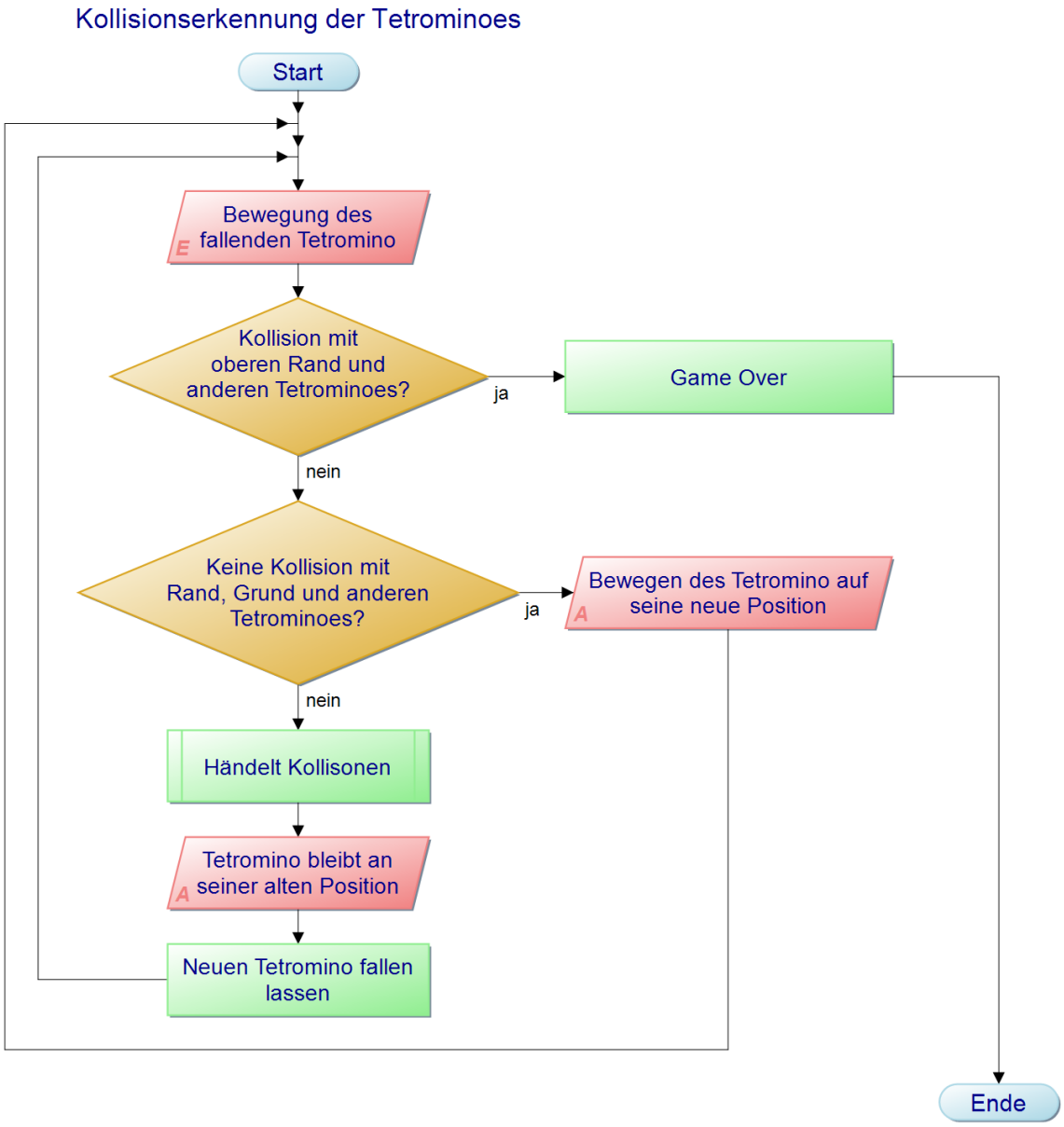


Abbildung 5: Kollisionserkennung der Tetrominoes

Die Kollisionserkennung der Tetrominoes ist in Abbildung 4 veranschaulicht dargestellt. Wenn der Tetromino fällt wird geprüft, ob es eine Kollision am oberen Rand gibt. Trifft das zu, kommt das Game Over Fenster und das Spiel ist zu Ende und ein neues Spiel kann begonnen werden. Gibt es keine Kollision, fällt der Stein weiter und es wird geprüft, ob es keine Kollision mit einem anderen Tetromino gibt oder ob der Grund/Boden des Spielfeldes erreicht wurde. Trifft das zu fällt der Tetromino weiter und die Überprüfung fängt von vorne an. Trifft das nicht zu, dann ist eine Kollision vorhanden. Es wird nun die Kollision verarbeitet und der Tetromino bleibt an seiner alten Position. Die Folge daraus ist, dass ein neuer Tetromino fallengelassen wird und die Kollisionsüberprüfungen fangen erneut an.

3.1.3 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 Model
- _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.

3.1.4 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.

3.1.5 Goal

Ein Goal zeigt während des Spiels das Ziel des aktuellen Levels an. Der Zustand eines Goals besteht aus

- _level das zum Ziel gehörige Level
- _description Beschreibung des Ziels
- _goalValue Wert um das Ziel zu erfüllen

Die Goal Klasse hat folgende Methoden:

- Goal() Konstruktor der Goal Klasse
- getProgress() hält den Fortschritt des Ziels fest
- isCompleted() prüft ob die für dieses Level spezifizierte Ziel erfüllt wurde

Um ein neues Ziel zu erstellen, muss man eine neue Klasse erstellen, die von der abstrakten Goal Klasse erbt. Damit ist man gezwungen, die in der abstrakten Goal Klasse enthaltenen Methoden zu implementieren. Als Beispiel, dienen die drei Klassen die bereits existieren, dabei handelt es sich um EndlessGoal, NumberOfTetrominoesFallenGoal und NumberOfRowClearedGoal. Die Klassen müssen immer getProgress() und isCompleted() implementieren, damit man den Fortschritt des Ziels und das Ende des Ziels kennt. Das Ziel muss dann in der Json Datei dem Level zugeordnet werden.

3.1.6 PowerUp

Ein PoweUp kann Aspekte des Spiels manipulieren, wenn es aktiviert wird. Der Versuch ein PowerUp zu aktivieren kann an fast allen Stellen des Spiels passieren. Ob das PowerUp zu diesem Zeitpunkt wirklich genutzt werden kann, muss in _isConsumable geprüft werden. Der Zustand eines PowerUps besteht aus

- _model das zum PowerUp zugehörige Model

- _id wird genutzt, um in serialisierten Objekt dieses PowerUp zu referenzieren. (zB. kann in der JSON Datei eines Tetrominos diese id angegeben werden, damit dieser Tetromino dieses PowerUp besitzt)

- getDescription gibt die Beschreibung des PowerUps zurück

Die PowerUp Klasse hat folgende Methoden:

- PowerUp() muss wenigstens das Model kennen, welches es manipulieren soll.

- Consume() aktiviert das PowerUp. Jedes PowerUp muss eine individuelle Implementation vornehmen.

- _isConsumable() implementiert die Bedingung, welche zur Aktivierung des PowerUps notwendig ist. kwargs enthält Daten, die benötigt werden, um zu überprüfen ob die Bedingung erfüllt wurde.

Um ein neues Power Up zu erstellen, muss man eine neue Klasse erstellen, die von der abstrakten PowerUp Klasse erbt. Damit ist man gezwungen, die in der abstrakten PowerUp Klasse enthaltenen Methoden zu implementieren. Als Beispiel dienen die Klassen die bereits existieren, dabei handelt es sich um die RemoveAllRowsOfTetromino Klasse und die TetrominoBomb Klasse. Die Klasse muss immer _isConsumable und Consume implementieren, damit das jeweilige Power Up und die notwendige Bedingung zum auslösen des PowerUps vorhanden ist. Das Power Up muss dann in der Json Datei dem jeweiligen Tetromino zugeordnet werden.

3.2 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).

3.2.1 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  <!DOCTYPE html>
2  <html lang="de">
3  <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="user-scalable=no, width=device-width,
```

```
6   initial-scale=1.0">
7       <title>Tetris</title>
8       <link href="favicon.ico" rel="shortcut icon" type="images/x-icon">
9       <link rel="stylesheet" type="text/css" href="styles.css">
10  </head>
11  <body>
12  <!-- Container für die Startseite -->
13  <div class="container_start">
14      <div id="title"></div>
15      <p id="start">START</p>
16      <p id="copy">&copy; Florian Jeger, Christoph Jeger &amp;amp; Matthias
17  Steffen</p>
18  </div>
19
20  <div id="overlay">
21  </div>
22
23  <!-- Container für die Nachrichten -->
24  <div class="container_message">
25      <img id="logo" src='img/tetris_menu_logo.png'>
26      <img id="portrait" src='img/portrait.png'>
27      <p id="message"></p>
28      <p id="continue">FORTSETZEN</p>
29      <p id="newGame">NEUES SPIEL</p>
30  </div>
31
32  <div class="container_powerup">
33      <p><b>PowerUp</b>: Dieser Tetromino löscht alle Reihen um sich
34  herum!</p>
35  </div>
36
37  <div class="container_game">
38      <!-- Container für die Level Ziele -->
39      <div class="container_goal">
40          <b>Ziel:</b> <span id="goalDescription"></span> <span
41  id="goalProgress">0</span>/<span id="goal">0</span>
42          <br/>
43          <b>Belonung: </b> <span id="bonusPoints">0</span> Punkte
44      </div>
45
46      <!-- Tabelle für das Spielfeld -->
47      <table id="field">
48      </table>
49
50      <div class="container_sidebox">
51          <!-- Container für die Seitenbox -->
52          <div class="container_nextstone">
53              <h3>Nächster Stein</h3>
54              <!-- Tabelle für den nächsten Stein -->
55              <table id="nextstone">
56              </table>
57          </div>
58
59          <div class="container_holdstone">
60              <h3>Stein halten</h3>
61              <!-- Tabelle für den nächsten Stein -->
62              <table id="holdstone">
63              </table>
64          </div>
65
66          <!-- Container für die Spielinformationen -->
67          <div class="container_info">
```

```
68              <h3>Level</h3>
69              <p id="level">0</p>
70              <h3>Punkte</h3>
71              <p id="points">0</p>
72          </div>
73      </div>
74
75
76      <!-- Container für die Steuerung -->
77      <div class="container_control">
78          <div id="menu"><img src='img/menu.png'></div>
79          <div id="hard_drop"><img src='img/hard_drop.png'></div>
80          <div id="left_rotation"><img src='img/left_rotation.png'></div>
81          <div id="right_rotation"><img src='img/right_rotation.png'></div>
82          <div id="hold"><img src='img/hold.png'></div>
83          <div id="left"><img src='img/left.png'></div>
84          <div id="down"><img src='img/down.png'></div>
85          <div id="right"><img src='img/right.png'></div>
86      </div>
87
88      <script type="application/dart" src="tetrisclient.dart"></script>
89      <script data-pub-inline src="packages/browser/dart.js"></script>
90  </div>
91  </body>
92  </html>
93
```

Listing 1: HTML Basisdokument des Spiels

Um das Spiel einzublenden wird diese HTML-Dokument genutzt.

3.2.2 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.

3.3 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.

4. Level- und Parametrisierungskonzept

4.1 Levelkonzept

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.

Die Fallgeschwindigkeit und die Bonuspunkte lassen sich in der JSON-Datei (Ordner web) anpassen pro Level, siehe Abschnitt 4.2.3.

4.2 Parametrisierungkonzept

Das Spiel, dessen Level und die Tetrominoes lassen sich über eine JSON-Datei parametrisieren. Um diese Funktionalität zu implementieren war es notwendig alle Eigenschaften die eine Entität definieren serialisierbar zu machen und eine Struktur zu definieren, die es ermöglicht diese Eigenschaften programmatisch auszulesen und entsprechende Entitäten zu erstellen. Im Folgenden soll erläutert werden, wie diese Struktur aussieht. Dies ermöglicht auch Nutzer mit geringen technischen Fähigkeiten das Spiel um neue Level oder Tetrominoes zu erweitern.

Alle Entitäten, welche in der JSON-Datei konfiguriert werden, müssen eine eindeutige Id besitzen. Über diese Id können in der JSON-Datei Referenzen zwischen Entitäten realisiert werden. Zu finden ist die game-config.json Datei im Ordner web.

4.2.1 Allgemeine Konfiguration

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

Listing 2: Parametrisierung des Spielfeldes

Unter Listing 2 wird die Parametrisierung des Spielfeldes angeben. Der Schlüssel ist gameConfiguration der genau einmal vorkommen muss. Die Attribute sind fieldWidth (default: 10) und fieldHeight (default: 18) wo die Breite und Höhe des Spielfeldes anzugeben ist.

4.2.2 Tetrominoes

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

Listing 3: Beispiel Parametrisierung eines Tetrominoes

Um ein Tetromino zu erstellen muss eine Parametrisierung vorgenommen werden, siehe Listing 3. Schlüssel tetrominoes enthält eine List von Objekten die Tetrominoes beschreiben, alle Tetrominoes die verwendet werden können, müssen hier definiert werden. Unter id wird der Name des Tetromino angegeben (eindeutige id für dieses Objekt), 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 angegeben werden. Genau beschreibt stones die initiale Position des Tetrominos, sobald er zu Spielfeld hinzugefügt wird. Die Position aller Bestandteile dieses Tetrominos wird horizontal relativ zur Mitte des Spielfeldes wie folgt angegeben: {„row“: horizontaler Offset (int), „col“: vertikaler Offset (int)}

Bei Preview ist es genauso wie stones nur für die Vorschau Box. Daher beschreibt preview ein Tetromino der in der Vorschau Box angezeigt werden soll. Für jeden Bestandteil des Tetrominos wird dessen Position innerhalb der Vorschau Box wie folgt angegeben: {„row“: Reihe (int), „col“: Spalte (int)}

Für die Drehung des Tetromino muss eine Dreh Matrix angeben werden unter transitions, siehe Abbildung 3.

Bei powerUps wird angeben, ob der Stein ein Power Up hat oder nicht. Dazu werden die Ids der Powerups angegeben. Verfügbare Powerups sind: „RemoveAllRowsOfTetromino“

Die Farbe des Tetromino muss unter color angegeben werden. Verfügbare Farben sind: „cyan“, „blue“, „yellow“, „orange“, „red“, „green“, „purple“

4.2.3 Level

```
{
  "id": "level2",
  "availibleTetrominoes": [
    "BombTetromino",
    "ITetromino",
    "TTetromino",
    "JTetromino",
    "LTetromino",
    "OTetromino",
    "STetromino",
    "ZTetromino"
  ],
  "scoreMultiplier": 2.0,
  "tetrominoSpeedInMs": 800,
  "goal": {
    "numberOfRowsCleared": 4.0
  },
  "bounsPoints": 1250,
  "priority": 80
},
```

Listing 4: Beispiel Parametrisierung von einem Level

Im Listing 4 wird anhand des Level 2 beispielhaft gezeigt wie ein Level parametrisiert wird. Die id gibt an, um welches Level es sich handelt (eindeutige id für dieses Objekt). Die Tetrominoes die in dem Level verfügbar sein sollen werden unter availibleTetrominoes angegeben. Somit enthält es alle Tetromino ids, die in diesem Level verfügbar sein sollen. Der scoreMultiplier gibt an um wie viel die Punkte multipliziert werden soll, in dem jeweiligen Level. TetrominoSpeedInMs Beschreibt die Zeit (in ms) die zwischen zwei Fallbewegungen eines Tetrominoes vergeht.

Das Ziel, welches in diesem Level erfüllt werden soll beschreibt goal. Das Objekt besteht aus einem Key-Value Paar. Der Schlüssel gibt über die Id (String) das zu erfüllende Ziel an und

der zugehörige Wert (double) quantifiziert das Ziel. Ein Beispiel: {„*numberOfRowsCleared*“: 3.0} legt fest, dass der Spieler 3 Tetromino Reihen lösen muss, um das Level zu beenden. Verfügbare Ids für Ziele sind:

„numberOfRowsCleared“, "numberOfTetrominoesFallen"

Unter bonusPoints werden die Bonuspunkte für das erfolgreiche beenden des Levels festgelegt.

Die Priorität dieses Levels legt priority fest. Sind mehrere Level vorhanden wird anhand dieser, deren Reihenfolge bestimmt. Das Level mit der höchsten Priorität wird dabei als erstes gespielt.

5. 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.

5.1 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	x			Das Levelkonzept ist vorhanden und wird im Abschnitt 4.1 dargestellt.
AF-4	Parametrisierungs-konzept	x			Das Parametrisierungskonzept ist vorhanden und wird im Abschnitt 4.2 dargestellt.
AF-5	Mobile Browser		x		Das Spiel funktioniert im Browser Chromium/Dartium. Das Spiel muss ferner in allen anderen mobile Browsern funktionieren. Geprüft wurden Opera, Microsoft Edge, Chrome und Firefox. Leider funktioniert das Spiel im Safari Browser auf dem Smartphone nicht vollständig, weil Apple mit der iOS 10 Version es nicht mehr zulässt, das im Safari Browser die Doppel Tipp Zoomfunktion ignoriert werden kann.

Tabelle 3: Nachweis der funktionalen Anforderungen

5.2 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 dient der 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. Es werden nur die zugelassenen Pakete genutzt.

Tabelle 4: Nachweis der Dokumentationsanforderungen

5.3 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	x			Ein Levelformat ist vorhanden und lässt sich leicht in der game-config.json Datei (web Ordner) ändern oder hinzufügen.
TF-3	Parameterformat	x			Die Parametrisierung ist vorhanden und lässt sich leicht in der game-config.json Datei (web Ordner) ändern oder hinzufügen.
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 mobile Browsern funktionieren. Geprüft wurden Opera, Microsoft Edge, Chrome und Firefox. Leider funktioniert das Spiel im Safari Browser auf dem Smartphone nicht

					vollständig, weil Apple mit der iOS 10 Version es nicht mehr zulässt, das im Safari Browser die Doppel Tipp Zoomfunktion ignoriert werden kann. Bei der JavaScript kompilierten Form trifft der gleiche Sachverhalt wie oben beschrieben auf.
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

5.4 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			V	
	Tetris Spiel	/lib/src/model/ TetrisGame.dart	U		V	
	Tetromino	/lib/src/model/ Tetromino.dart	V			
	Level	/lib/src/model/ Level.dart	U		V	
	Ziele	/lib/src/model/ goals/*	U		V	
	Power Ups	/lib/src/model/ powerUps/*			V	
View	HTML-Dokument	web/index.html		V		

	Gestaltung	web/styles.css	V	U		
	Bilder	web/img/*	V	U		
	Viewlogik	/lib/src/view/ TetrisView.dart		V		
Controller	Eventhandling	lib/src/controller/ TetrisController.dart		V		
Dokumentation	Tetris-Documentation	doc/*.*	U	V	U	
	UML Diagramm	doc/Tetris Klassendiagramm.jpg		V		
Sonstiges	Json Datei	web/game-config.json	V		U	
Util	Builder & readers	lib\src\util*			V	

Tabelle 6: Projektverantwortlichkeiten

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

U = unterstützend (Übernahme von Teilaufgaben)