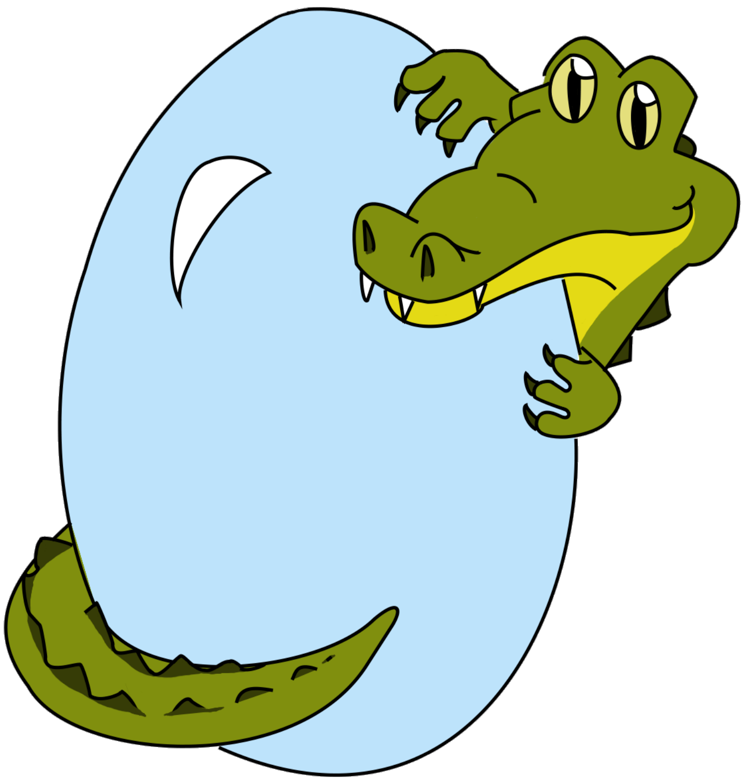
Croggle

Lernanwendung für Grundschüler

Entwurf

Lukas Böhm, Tobias Hornberger, Jonas Mehlhaus,
Iris Mehrbrodt, Vincent Schüßler, Lena Winter

23. Dezember 2013



Inhaltsverzeichnis

1 Einführung

Die Applikation „Croggle“ soll Grundschülern schon früh grundlegende Prinzipien und Funktionsweisen der funktionalen Programmierung, speziell dem untypisiertem Lambda-Kalkül, vermitteln. In diesem Entwurfsdokument werden sowohl die Designkonzepte, als auch die dahinterstehende Architektur der Applikation dargelegt.

Den wichtigsten Teil des Dokumentes bildet die Klassendokumentation, in der alle Klassen und ihre Methoden aufgelistet und beschrieben sind. Dies wird ergänzt durch das Klassendiagramm, das sowohl Attribute und Methoden, als auch die Beziehungen der Klassen untereinander darstellt. Weiterhin werden typische Programmabläufe, und die Interaktion der Klassen dabei, durch Sequenzdiagramme beschrieben. Außerdem wird auf andere Punkte, die für den Aufbau der Applikation wichtig sind, genau eingegangen. Dazu gehören der Aufbau der JSON Dateien für Level und Wunschkriterien, die im Pflichtenheft beschrieben, aber im Entwurf nicht umgesetzt wurden.

2 Designentscheidungen

2.1 Model View Controller

Um einen flexiblen Programmentwurf zu gewährleisten, wird das Architekturmuster „Model View Controller“ verwendet. Dieses soll spätere Änderungen und Erweiterungen erleichtern, sowie eine Wiederverwendung der einzelnen Programmkomponenten ermöglichen.

2.2 libgdx

Für die Entwicklung der App wird das Framework libgdx verwendet. Es ist speziell für Spielentwicklung ausgelegt und erfüllt damit die Anforderungen besser als das Android Application Framework. Libgdx bietet dem Benutzer die Möglichkeit, einfach ästhetisch ansprechende UIs aufzubauen und liefert von Haus aus vereinfachtes Rendering. Die umfangreiche Dokumentation erleichtert die Einarbeitung in das Framework. Auch die Tatsache, dass libgdx eine freie Software ist, spricht für die Nutzung.

2.3 Aufbau des „Model“

Im „Model“ werden die Alligatorenkonstellationen durch eine veränderbare (mutable) Baumstruktur repräsentiert. Im Gegensatz zur Variante, die Konstellationen in einer unveränderbaren (immutable) Form zu speichern, bietet diese Form den Vorteil, dass Referenzen auf Objekte eindeutig sind und in der Baumstruktur auch auf den Vorgänger (parent) eines Baumelements zugegriffen werden kann (getParent()). Muss die Konstellation im Laufe des Spiels geändert werden, können einzelne Teilbäume der Konstellation geändert werden. Allerdings ergibt sich hier das Problem, dass beim Einfügen von Teilbäumen an mehreren Stellen immer vollständige Kopien (deep copy) dieses Teilbaumes eingefügt werden müssen, damit keine Probleme mit Referenzen auftreten.

2.4 Visitor Pattern

Durch die Darstellung der Alligatorenkonstellation in einer Baumstruktur liegt die Verwendung des Visitor-Patterns nahe. Das Muster wird hier vorrangig benutzt um neue Operationen, die auf der Konstellation ausgeführt werden sollen, einfach hinzufügen zu können. Desweiteren sollen die verwandten Operationen, die auf den verschiedenen Objekten des Baumes ausgeführt werden, zentral im Besucher gespeichert werden. Dies geschieht mit der Intention, dass sie einfach und zentral verwaltet beziehungsweise modifiziert werden können.

3 Nicht umgesetzte Wunschkriterien

3.1 Sandboxlevel speichern/laden

Der Leveleditor wird wegen der sehr aufwändigen Umsetzung und daraus resultierender Zeitprobleme nicht in den Entwurf aufgenommen und daher aus dem Projekt gestrichen. Dies betrifft Punkt /F260+/ des Pflichtenheftes.

3.2 Avatarerstellung

Die Erstellung eigener Avatare durch den Nutzer wird nicht umgesetzt, da es für das eigentliche Kernspiel und das damit verbundene Lernen nicht von Relevanz ist. Es würde jedoch einen nicht zu vernachlässigenden Teil der verfügbaren Entwicklungs- und Implementatierungszeit in Anspruch nehmen, weshalb diese Entscheidung nötig wurde. Die Avatarwahl funktioniert stattdessen über die vom System bereitgestellten Avatare, die vom Spieler ausgewählt werden können. Dies betrifft den Punkt /F130+/ des Pflichtenheftes.

3.3 Automatische Kameraführung

Die automatische Kameraführung während der Auswertung wird nicht umgesetzt. Nach ausführlichem Abwägen wurde dieses Wunschkriterium wegen schlechtem Verhältnis von Nutzen zu Aufwand gestrichen. Dies betrifft Punkt /F236+/ des Pflichtenheftes.

3.4 Achievements

Die Achievements für einen nichtterminierenden Ausdruck und für eine bestimmte Anzahl an Zeilen und Spalten werden gestrichen. Der Grund dafür liegt darin, dass sich diese Achievements in der Umsetzung stark von den anderen Arten von Achievements unterscheiden. Wegen schwieriger Definition einer „Spielsitzung“ auf mobilen Geräten wird außerdem das Achievement für eine bestimmte Anzahl an gelösten Leveln in einer Spielsitzung gestrichen. Dies betrifft Teile des Punktes /F250+/ des Pflichtenheftes.

4 Level JSON Format Beschreibung

4.1 Präambel

Die Auslieferung von Levels des Spiels wird auf Basis des verbreiteten JSON (JavaScript Object Notation) Formates und mittels Konventionen auf der Dateisystemhierarchie geschehen. Die Gründe für die Verwendung von JSON sind

a) Einfache Schreib- und Editierbarkeit

b) Gute Les- und Nachvollziehbarkeit

c) Geringerer Overhead gegenüber XML

d) Unterstützung von Haus aus durch libgdx

e) Gute Austausch- und Erweiterbarkeit gegenüber datenbank- oder codebasierten Herangehensweisen

Dafür ist es zunächst jedoch erforderlich, dass die Struktur der letztendlich für den Aufbau eines Levels benötigten Daten, detailliert spezifiziert ist. Im folgenden werden die einzelnen Attribute eines Levels und ihre Bedeutungen, sowie ihre Repräsentation in JSON beschrieben.

4.2 Grundlegendes

Im Folgenden wird stets davon ausgegangen, dass sich jegliche Dateisystem Pfadangaben relativ zu „/assets“ beziehen, wobei als Wurzelverzeichnis das Android Projektverzeichnis angesehen wird. Die Nutzung des „asset“-Ordners wird hierbei von libgdx vorgegeben. Vergleiche hierzu den entsprechenden Artikel der Dokumentation [1]. Zudem sei erwähnt, dass sich alle in JSON angegebenen Elemente, die von der Anwendung benutzt werden, im Namespace **“de.croggle“** befinden. Das heißt, dass sich in dem Wurzelobjekt jeder

[1] `https://github.com/libgdx/libgdx/wiki/File-handling#android`

JSON Datei ein Objekt mit diesem Namen befindet. Dies macht es sicherer, gegebenenfalls in Zukunft vom Nutzer angegebene Levels zu laden, auch wenn dies derzeit nicht als Funktion vorgesehen ist.

Listing 4.1: Standardinhalt jeder JSON Datei der Anwendung

```
{
  "de.croggle" : {
    ...
  }
}
```

Die Spezifikation beschreibt lediglich die Elemente, die in Objekten verpflichtend vorhanden sein müssen. Darüber hinausgehende Attribute werden von der Applikation ignoriert. Sollten einzelne Elemente oder auch nur deren Inhalt als optional angesehen werden, ist dies entsprechend im Text ausgewiesen.

Da die Beschreibungen von Levels, ähnlich wie Quellcode, komplexe Sachverhalte im Lambda-Kalkül abbilden können, deren Wirkungsweise nicht direkt ersichtlich sein kann, sieht die Spezifikation der JSON Repräsentation Kommentarattribute vor. Diese können und sollen verwendet werden, um besonders schwer nachzuvollziehende Stellen zu dokumentieren, aber auch, um einfach auf die Ideen und Zielsetzungen innerhalb eines bestimmten Levels hinzuweisen. Solche Kommentarattribute werden durch das Präfix **“_comment“** markiert und eingeleitet. Attributsnamen, die mit diesem beginnen sind dementsprechend reserviert. Es muss zudem davon ausgegangen werden, dass sich an einer späteren Stelle der Entwicklung ein Mechanismus im Build System befindet, der die Kommentare vor der Auslieferung aus den JSON Dateien entfernt.

Da die Kommentare jeweils durch eine Zeichenkette im Namen eines Attributes eingeleitet werden und JSON keine Namen von Attributen in Listen (eingeleitet durch “[“) vorsieht, sind Kommentare nur in Objekten (eingeleitet durch “{“) zulässig. Dies ist insofern keine Einschränkung, da Listen im Allgemeinen nur anonyme Instanzen von Elementen beinhalten, deren es für gewöhnlich keinerlei weitere Erklärungen bedarf. Der Wert eines solchen Kommentarattributes kann einerseits aus einem einfachen Stringliteral (Zeichenkette) bestehen, wenn es sich um kurze, einzeilige Kommentare handelt. Andererseits kann bei längeren Kommentaren auch eine Liste von Stringliteralen angegeben werden, wobei jedes Element auf eine neue Zeile geschrieben wird. Damit wird der Einschränkung im JSON Format Rechnung getragen, dass es keine Möglichkeit gibt, sogenannte Multiline Strings anzugeben. Zuletzt ermöglicht die Spezifikation auch mehrere Kommentarattribute innerhalb des gleichen Objektes. Dafür wird den jeweiligen **“_comment“**-Attributnamen noch jeweils eine Ziffernfolge angehängt, welche von Null beginnend aufsteigt. Dies ist nötig, da der JSON Standard keine gleichen Attributnamen innerhalb einunddesselben Objekts vorsieht. Listing 4.2 verdeutlicht die Anwendung von Kommenatren in JSON Files.

Listing 4.2: Kommentare in einer JSON Datei

```
{
  "_comment" : "Dies ist ein kurzer Kommentar",
  "de.croggle" : {
    "_comment": [
      "Längere Kommentare, die ",
      "über mehrere Zeilen gehen, ",
      "werden mittels einer Liste repräsentiert"
    ]
  },
  "_comment0" : "Mehrere Kommentare in ein und demselben Objekt...",
  "_comment1" : "... werden mittels aufsteigender Nummern als Postfix
      unterschieden"
}
```

4.3 Levels

Bisher sind folgende Leveltypen für das Spiel vorgesehen:

- Multiple Choice Levels (MC)
- Färben und Einfügen (FE)
- Schrittanzahl (SA)

Um ein Level zur Laufzeit zu laden, müssen die folgenden Daten zur Verfügung stehen:

a) Das Levelpaket, zu dem ein Level gehört

b) Die Position des Levels innerhalb eines Levelpakets

c) Eine Zeichenkette, die auf das Level einstimmt

d) Eine (optionale) Animation zu Beginn eines Levels

e) Eine optionale Anzahl Schritte, nach der die Simulation automatisch beendet wird

f) Der Name eines Designthemas (es kann sich auch nur um den Pfad zu einem Hintergrundbild handeln)

g) Ein Tipp, um das Level abzuschließen

h) Den Typ eines Levels

i) Konstellationen (Anfang, Ende, mögliche Antworten)

Zur Angabe dieser Daten existiert zu jedem Level eine JSON Datei (Endung **“.json“**), deren Name der auf zwei Stellen aufgefüllten Identifikationsnummer innerhalb eines Levelpakets entspricht. Dies ermöglicht das schnelle Auflisten aller Levels eines Pakets, impliziert aber auch, dass ein Levelpaket höchstens 100 Level enthalten kann. Zur Vermeidung von Namenskollisionen zwischen Levels verschiedener Pakete, erhält jedes Paket sein eigenes Unterverzeichnis. Dazu jedoch mehr in Kapitel “Levelpakete“. Innerhalb einer solchen JSON Datei existiert (im Applikationsnamensraum) eine Liste mit dem Namen **“levels“**. An dieser Stelle wäre auch ein einfaches Objekt möglich, welches direkt alle notwendigen Daten enthält. Der gewählte Ansatz bietet allerdings die Vorteile, dass erstens direkt auf die Levels als solche zugegriffen werden kann, da keine Typprüfung stattfinden muss (**“levels“** darf nur Levels enthalten). Zweitens lassen sich gegebenenfalls alle Levels in dieser Liste zusammenlegen, was nur noch eine JSON Datei notwendig machen und somit Spielraum für Leistungsverbesserungen bieten würde. Und drittens muss kein Name für die sonst eigentlich anonymen Levels angegeben werden.

Innerhalb dieses Grundgerüstes werden alle weiteren, notwendigen Daten spezifiziert. Zuerst genannt sei hierbei die Beschreibung: Unter dem Attributsnamen **“description“** wird hierbei die auf das Level einstimmende Zeichenkette abgelegt. Unter dem Attribut **“design“** ist, wie erwähnt, entweder der Name eines Designthemas oder der Name/Pfad zu einem alternativen Hintergrundbild in einer Zeichenkette gespeichert. Bei einer leeren Zeichenkette wird das Standardthema des Levelpakets verwendet. In einem Feld **“abort simulation after“** kann optional die ganzzahlige Anzahl an Ausführungsschritten abgelegt werden, nach denen ein Level als beendet angesehen wird. Positive Werte implizieren, dass im Anschluss das Level als erfolgreich abgschlossen gewertet werden soll. Bei einem negativen Wert gilt das Level nach Ablauf der Schrittzahl als verloren. Der Wert Null (“0“) bedeutet hierbei, dass kein Abbruch geschieht, was dem voreingestellten Verhalten entspricht. Das Attribut **“hints“** bezeichnet eine Liste von Zeichenketten, die Pfade zu Hilfegrafiken angeben, die der Nutzer als Hilfestellung anzeigen lassen kann. Im derzeitigen Entwurf ist nur ein einziger Tipp pro Level vorgesenen. Orientiert an anderen Spielen auf dem Markt hält eine Liste allerdings die Möglichkeit offen, weitere Tipps pro Level hinzuzufügen, ohne dass die JSON Spezifikation geändert werden muss. In diesem Falle würden die Tipps in der Reihenfolge aufgedeckt werden, in der sie in der Liste vorkommen. Die Liste an sich kann allerdings auch leer bleiben.

Unter dem Namen **“type“** muss eine Zeichenkette existieren, die einen der folgenden Werte enthalten muss:

- multiple choice
- modification
- step count

Für jeden dieser Typen unterscheidet sich der Inhalt des letzten Attributs, dem **“data“** Attribut. Dieses enthält die für die einzelnen Typen unterschiedlichen Daten. Diese werden im Folgenden spezifiziert. Listing 4.3 beschreibt das benötigte Rahmenwerk für alle Levels.

Listing 4.3: JSON Leveldatei, z.B. json/00/00.json

```
{
  "de.croggle" : {
    "levels" : [
      {
        "type" : "...",
        "description" : "...",
        "design" : "...",
        "abort simulation after": "...",
        "hints" : [
          ...
        ],
        "data" : {
          ...
        }
      }
    ]
  }
}
```

4.3.1 Darstellung von Lambda-Ausdrücken

Alle Leveltypen haben gemein, dass sie an mindestens einer Stelle ein Spielfeld mit einem Alligatorausdruck anzeigen. Daher wird zunächst auf die Darstellung eines solchen Spielfelds mit der sich darauf befindlichen Konstellation (dem Lambda-Term) in der JSON Levelbeschreibung eingegangen. Diese Darstellung bildet das zu den Lambda-Termen gehörige Klassenmodell transparent ab.

Das heißt, dass Spielfelder (im Folgenden auch Boards genannt) eine Liste von Elementen, genauer gesagt von BoardObjects, als Attribut besitzen. Von diesen BoardObjects existieren drei Typen, nämlich Ei, farbiger Alligator und alter Alligator. Alle diese drei Objekttypen verfügen über eine Zeichenkette mit Namen **“type“**. Zusätzlich wird über den Wahrheitswert im Feld **“movable“** festgelegt, ob der Nutzer das Objekt nachträglich bewegen darf. Ob er es ganz vom Spielfeld entfernen darf, wird mit dem Feld **“removable“** angegeben.

Das **“type“**-Attribut, darf die folgenden Werte annehmen:

- egg

- colored alligator
- aged alligator

Aus diesem Attribut wird der genaue Typ eines BoardObjects abgeleitet, aus dem die weiteren Attribute hervorgehen. Diese sind im Folgenden beschrieben.

Eier

Objekte vom Typ **“egg“** besitzen die folgenden Eigenschaften:

color: Eine Ganzzahl >= 0, die einer Farbe zugeordnet wird. Der Wert befindet sich zwischen 0 und 30. Negative Werte werden als “keine Farbe/ungefärbt“ interpretiert.

recolorable: Wahrheitswert, ob das Objekt vom Nutzer nachträglich manuell umgefärbt werden darf.

Farbige Alligatoren

Objekte vom Typ **“colored alligator“** besitzen die folgenden Eigenschaften:

color: Eine Ganzzahl >= 0, die einer Farbe zugeordnet wird. Der Wert befindet sich zwischen 0 und 30. Negative Werte werden als “keine Farbe/ungefärbt“ interpretiert.

recolorable: Wahrheitswert, ob das Objekt vom Nutzer nachträglich manuell umgefärbt werden darf.

children: Eine Liste von weiteren BoardObjects

Alte Alligatoren

Objekte vom Typ **“aged alligator“** besitzen die folgenden Eigenschaften:

children: Eine Liste von weiteren BoardObjects

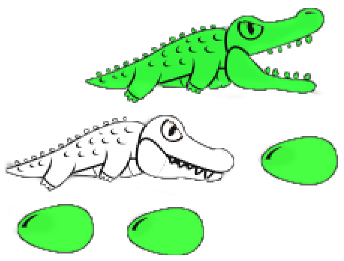
Listing 4.4: Ein einfaches Board mit allen existierenden BoardObjects

```json
{
  "families" : [
    {
      "type" : "colored alligator",
      "movable" : false,
      "removable" : false,
      "color" : 0,
      "recolorable" : false,
      "children" : [
        {
          "type" : "aged alligator",
          "movable" : false,
          "removable" : false,
          "children" : [
            {
              "type" : "egg",
              "movable" : false,
              "removable" : false,
              "color" : 0,
              "recolorable" : false
            },
            {
              "type" : "egg",
              "movable" : false,
              "removable" : false,
              "color" : 0,
              "recolorable" : false
            }
          ]
        },
        {
          "type" : "egg",
          "movable" : false,
          "removable" : false,
          "color" : 0,
          "recolorable" : false
        }
      ]
    }
  ]
}
```

4.3.2 “Multiple Choice“ Leveldaten

Multiple Choice Levels benötigen eine Spielfeldkonstellation für die Ausgangsstellung und eine Liste mit möglichen Antwortoptionen, die auch als Spielfelder modelliert werden. Hierbei wird bewusst keine feste Anzahl an Antworten festgelegt, auch wenn die Unterstützung durch das Programm nach der Implementierung nur bestimmte Anzah-

Abbildung 4.1: Der durch das vorausgehende Listing dargestellte Alligatorausdruck



len vorsieht. Außerdem wird noch ein Index für die Konstellation benötigt, welche die Richtige ist. Die Ausgangsstellung wird als Board in einem Feld mit Namen **“initial constellation“** angegeben. Die Liste mit Boards, die als mögliche Antworten dienen, heißt **“answers’"**. Diese Liste muss mindestens ein Element enthalten. Der Index der richtigen Antwort wird als Ganzzahl im Schlüssel **“correct answer“** gespeichert. Er bezieht sich auf die Position der gemeinten Konstellation in der **“answers“** Liste, wobei ab Null angefangen wird zu indizieren. Es ist jedoch zu beachten, dass die Reihenfolge der Elemente in der Liste keinen Einfluss auf deren Position in der dem Nutzer gezeigten Liste mit Antworten hat. Diese wird nämlich randomisiert erstellt.

Listing 4.5: Grober Aufbau des data Attributs eines Multiple Choice Levels

```
"type" : "multiple choice",
"data" : {
  "initial constellation" : {
    ...
  },
  "answers" : [
    ...
  ],
  "correct answer" : 0
}
```

4.3.3 “Färben und Einfügen“ Leveldaten

Bei “Färben und Einfügen“-Levels werden nur die Ausgangskonstellation, die zu erreichende Endkonstellation und Listen mit für den Nutzer gesperrte Farben und Objekttypen als Angabe benötigt. Die Ausgangskonstellation findet sich wie bei Multiple Choice Leveln im Feld **“initial constellation“**, und wird mittels eines Board Objektes repräsentiert. Die Endkonstellation kann über das Board-wertige Attribut **“objective“** ausgelesen werden. Gesperrte Farben werden durch eine Liste mit Ganzzahlen >= 0 dargestellt, die den Namen **“blocked colors“** trägt. Gesperrte Objekttypen können in einer Liste von Zeichenketten mit Namen **“blocked types“** angegeben werden, wobei deren

Werte den bekannten Typen aus der Dartellung von Lambda-Ausdrücken entspricht. Diese Funktion ist vor allem für die Tutorial Levels vorgesehen, wenn das Platzieren von Eiern etc. verhindert werden soll.

Listing 4.6: Grober Aufbau des data Attributs eines “Färben und Einfügen“ Levels

```
"type" : "modification",
"data" : {
  "initial constellation" : {
    ...
  },
  "objective" : {
    ...
  },
  "blocked colors" :  [
    ...
  ],
  "blocked types" : [
    ...
  ]
}
```

4.3.4 “Schrittanzahl“ Leveldaten

In Levels, in denen die Schrittanzahl zum Gewinnen des Levels entscheidend ist, werden grundsätzlich die gleichen Daten im data Attribut bereitgestellt, wie schon für den Typ “Färben und Einfügen“. Lediglich ist hier das Element **“objective“** nicht mehr vonnöten. Das Ziel wird stattdessen nur von dem ohnehin für jeden Leveltyp vorgesehenen Wert **“abort simulation after“** außerhalb des data Attributs spezifiziert.

Listing 4.7: Grober Aufbau des data Attributs eines “Schrittzahl“ Levels

```
"type" : "step count",
"data" : {
  "initial constellation" : {
    ...
  },
  "blocked colors" :  [
    ...
  ],
  "blocked types" : [
    ...
  ]
}
```

4.4 Levelpakete

Levels werden durch sogenannte Pakete gruppiert. Diese Pakete besitzen folgende Eigenschaften, die durch die Spezifikation repräsentiert werden müssen:

a) Eine Position des Pakets, um die Reihenfolge der Pakete festzulegen

b) Eine Verbindung zwischen dem Paket und den dazugehörigen Leveln

c) Ein Name zur Bezeichnung des Pakets

d) Eine kurze, optionale Beschreibung des Pakets

e) Eine Grafik, die zum Thema des Pakets passt

f) Ein Designthema, in dem das Paket und die enthaltenden Levels standardmäßig angezeigt werden.

g) Eine Angabe zu etwaigen Animationen, die beim ersten Spielen des Pakets gezeigt wird.

h) Abhängigkeiten, die zum Freischalten der Box erfüllt sein müssen

Hierbei muss jedoch nicht allein auf JSON zurückgegriffen werden. Auch Dateisysteminformationen können berücksichtigt werden. Es muss lediglich bedacht werden, dass durch Festlegung von Konventionen keine zu starke Einschränkung passiert, sodass die Erweiterbarkeit nicht eingeschränkt wird.

Grundsätzlich existiert zu jedem Level unter json/levels/ ein Ordner mit der Position des Pakets in der Liste der Pakete. Auch hier wird der Index als zweistellige Zahl mit den Ziffern 0 bis 9 dargestellt, was eine theroetische maximale Anzahl von 100 verschiedenen Levelpaketen erlaubt. In jedem dieser Ordner befinden sich die zu dem Paket gehörenden Levels in jeweils einer eigenen JSON Datei, die nach dem in Abschnitt "Levels" beschriebenen Schema benannt sind. Diese Maßnahmen decken bereits die ersten beiden Punkte, die für die Beschreibung eines Levelpakets nötig sind, ab. Zusätzlich existiert in dem Paketordner eine weitere JSON Datei mit dem Namen **"package.json"**. Diese dient dazu, die restlichen der eben aufgelisteten Daten anzugeben. Wie dies genau geschieht wird im Folgenden spezifiziert.
Wie bereits bei Levels geschieht die Definition eines Levels nicht über einen festen Namen im Namensbereich des Spiels. Stattdessen wird ein anonymes Objekt in einer Liste verwendet. Diese Liste heißt **"packages"**. Listing 4.8 verdeutlicht die Folgen dieses Sachverhalts. Der Name eines Packages wird im Feld **"name"** als Zeichenkette eingetragen. Die Zeichenkette zur Beschreibung erfolgt wie bei Leveln im **"description"** Attribut.

Mittels einer Zeichenkette **“banner“** wird der Pfad zu einer für das Level stehenden Grafik spezifiziert. Auch die Angaben von Designthema und Animation beim ersten Betreten werden durch Pfade in Zeichenketten angegeben. Für ersteres heißt diese **“design“**, für zweiteres **“animation“**. Die Paketdesigns können von Levels mittels des dazu vorgesehenen **“design“** Attributes überschrieben werden. Da die Beschreibung von Abhängigkeiten eine größere Komplexität aufweist, ist ihr im folgenden eine Untersektion gewidmet. Grundsätzlich werden sie in einer Liste mit dem Namen **“dependencies“** definiert. Das folgende Listing veranschaulicht all diese Spezifikationen.

Listing 4.8: Standardinhalt der Definition eines Levels

```
{
  "de.croggle" : {
    "packages" : [
      {
        "_comment" : "Die Definition eines Paketes erfolgt hier",
        "name" : "...",
        "decription" : "...",
        "banner" : "...",
        "design" : "...",
        "animation" : "...",
        "dependencies" : [
          ...
        ]
      }
    ]
  }
}
```

4.4.1 Abhängigkeiten

Zum Freischalten eines Levelpakets können folgende Typen von Abhängigkeiten vorgegeben sein:

- Vorhergehendes Levelpaket zu einem gewissen Grad abgeschlossen

Abhängigkeiten werden grundsätzlich durch Objekte abgebildet, die mindestens das Attribut **“type“** aufweisen, mit dem weitere Felder - je nach Typ - eingeführt werden können. Folgende Typen sind bisher definiert:

packageprogress: Wie weit ein bestimmtes anderes Levelpaket abgeschlossen sein muss, damit das Paket freigeschaltet ist.

Im Folgenden sind den jeweiligen Typen Abschnitte gewidmet, in denen weitere Attribute, die durch die Angabe eines Typseingeführt werden, erklärt sind.

Paketfortschritt: “packageprogress“

Für die Angabe eines mindestens benötigten Fortschritts eines Pakets werden Felder für das gemeinte Paket und für den Fortschritt benötigt. Für Ersteres heißt dieses **“package“** und beinhaltet eine Ganzzahl größer Null und kleiner 100. Zweiteres findet sich im Feld **“progress“**, welches auch eine Ganzzahl enthält, die den Fortschritt in Prozent darstellt. Der Wert befindet sich also zwischen einschließlich 0 und 100. Null Prozent heißt hierbei lediglich, dass das Level freigeschaltet sein muss. Das Gesperrtsein eines Pakets kann also keine Abhängigkeit sein. Bei hundert Prozent müssen alle Levels erfolgreich abgeschlossen worden sein, die zu dem angegebenen Paket gehören. Das folgende Listing fasst dies noch einmal zusammen.

Listing 4.9: Beispielhafte Definition einer Abhängigkeit vom Typ “boxprogress“

```json
{
  "type" : "packageprogress",
  "_comment" : "Alle Levels des ersten Pakets müssen zum Freischalten
      erfolgreich abgeschlossen worden sein",
  "package" : 0,
  "progress" : 100
}
```

4.5 Zusammenfassung

4.5.1 Dateisystemstruktur

Ordnerübersicht zur Levelspezifikation

```
assets/
└─json/
   └─levels/
      ├─00/
      │  ├─00.json
      │  ├─01.json
      │  ├─...
      │  └─package.json
      └─01/
         └─...
```

4.5.2 Inhalte einer „package.json“ Datei

```
"de.croggle" : Object - Namensraum
```

- **"packages"** : List - Liste mit Repräsentationen von Paketen. Gewöhnlich nur ein Paket
  - : Object - Das zur Datei gehörende Paket als JSON Objekt
    - **"name"** : String - Der Anzeigename des Levelpakets
    - **"description"** : String - Eine Zeichenkette zur Beschreibung des Pakets
    - **"banner"** : String - Pfad zu einer Bilddatei, die als Banner für ein Levelpaket fungiert
    - **"design"** : String - Pfad zu einem Design, das auf alle Levels eines Pakets angewendet wird
    - **"animation"** : String - Pfad zu einer Animation, die vor dem ersten Betreten des Levels abgespielt wird
    - **"dependencies"** : List - Die Liste mit Abhängigkeiten
      - **"type"** : String - Der Typ einer Abhängigkeit
      - Weitere Elemente für "packageprogress" Abhängigkeiten
        - **"package"** : unsigned Integer - Die Identifikationsnummer eines Pakets als Abhängigkeit
        - **"progress"** : unsigned Integer - Prozentzahl, zu der ein Paket als Abhängigkeit abgeschlossen sein muss

4.5.3 Inhalte einer Level JSON Datei

Rahmenwerk

- **"de.croggle"** : Object - Namensraum
  - **"levels"** : List - Liste mit Levelobjekten. Gewöhnlich nur ein Paket
    - : Object - Darstellung einzelner Levels
      - **"type"** : String - Typ eines Levels
      - **"description"** : String - Beschreibung zu einem Level
      - **"design"** : String - Pfad zu einem Leveldesign
      - **"abort simulation after"** : Integer - Nach wie vielen Schritten ein Level gewonnen (positiv) oder verloren (negativ) ist
      - **"hints"** : List - Strings mit Pfaden zu Hilfegrafiken
      - **"data"** (multiple choice) : Object - Spezielle Daten für die einzelnen Leveltypen
        - Daten für MC Levels .6 **"initial"** : Object - Board Objekt der Anfangskonstellation
          - **"answers"** : List - Liste mit Board Objekten der möglichen Antworten
          - **"correct answer"** : unsigned Integer - Der Index der richtigen Antwort zur Fragestellung
        - Daten für FE Levels
          - **"initial constellation"** : Object - Board Objekt der Anfangskonstellation

-
  - **"objective"** : Object - Board Objekt der zu erreichenden Konstellation
  - **"blocked colors"** : List - Liste mit Integer Werten, die blockierte Farben beschreiben
  - **"blocked types"** : List - String Liste mit Namen von Elementtypen, die im Level nicht platzierbar sind (egg, colored alligator, aged alligator)
- Daten für SA Levels
  - **"initial constellation"** : Object - Board Objekt der Anfangskonstellation
  - **"blocked colors"** : List - Liste mit Integer Werten, die blockierte Farben beschreiben
  - **"blocked types"** : List - String Liste mit Namen von Elementtypen, die im Level nicht platzierbar sind (egg, colored alligator, aged alligator)

Alligator-Baumstruktur

: Board - Objekt, das ein Spielfeld mit einer Konstellation modelliert
- **"families"** : List - Eine Liste mit sogenannten "BoardObjects"
  - BoardObject Objekte - Objekte, die Teil einer Konstellation auf dem Spielfeld sind
    - **"type"** : String - Der genaue Typ des vorliegenden BoardObjects. Die Unterscheidungen der Typen werden im Folgenden erläutert
    - **"movable"** : Bool - Ob das Objekt nachträglich vom Nutzer bewegt werden kann
    - **"removable"** : Bool - Ob das Objekt nachträglich vom Nutzer vom Board entfernt werden kann
    - Die drei möglichen Typen von BoardObjects sind "egg", "colored alligator" und "aged alligator"
      - Typ **"egg"**
        - **"color"** : Integer - Die Identifikationsnummer der Farbe, die das Ei haben soll
        - **"recolorable"** : Bool - Ob das Objekt nachträglich vom Nutzer umgefärbt werden kann
      - Typ **"colored alligator"**
        - **"color"** : Integer - Die Identifikationsnummer der Farbe, die der Alligator haben soll
        - **"recolorable"** : Bool - Ob der Alligator nachträglich vom Nutzer umgefärbt werden kann
        - **"children"** : List - Eine Liste mit weiteren BoardObjects (Eier, Alligatoren...), wie sie in diesem Listing beschrieben sind
      - Typ **"aged alligator"**
        - **"children"** : List - Eine Liste mit weiteren BoardObjects (Eier, Alligatoren...), wie sie in diesem Listing beschrieben sind

5 Level

In den Anfangskonstellationen werden ungefärbte Elemente durch die Variable **“o“** dargestellt. Besondere Level wie Tutoriallevel haben hier zusätzlich eine Beschreibung um ihr Ziel zu verdeutlichen. Farben werden hier als Variablen dargestellt.

5.1 Levelpacket 1

- Level 1

  Typ: Färbelevel

  Anfangskonstellation: o

  Endkonstellation: x

  gesperrte Farben: -

  Beschreibung: Erstes Tutoriallevel, in dem das Einfärben von Elementen erklärt wird.

- Level 2

  Typ: Färbelevel

  Anfangskonstellation: $(\lambda o.o)y$

  Endkonstellation: y

  gesperrte Farben: y

  Beschreibung: Zweites Tutoriallevel, in dem die β-Reduktion gezeigt wird. Benötigte Kenntnis des Spielers hierfür ist das Einfärben von Elementen.

- Level 3

  Typ: Einfügelevel

  Anfangskonstellation: $\lambda x.x$

  Endkonstellation: y

  gesperrte Farben: -

  Beschreibung: Drittes Tutoriallevel, in dem das Einfügen von Elementen auf das Spielfeld erklärt wird. Benötigte Kenntnis des Spielers hierfür ist das Einfärben von Elementen.

- Level 4

Typ: Einfügelevel

Anfangskonstellation: $\lambda x.x$

Endkonstellation: yy

gesperrte Farben: -

- Level 5

  Typ: Einfügelevel

  Anfangskonstellation: xyz

  Endkonstellation: xz

  gesperrte Farben: x, y, z

- Level 6

  Typ: Einfärbelevel mit Schrittanzahl

  Schrittanzahl: 5

  Anfangskonstellation: $\lambda o.o\lambda o.o\lambda o.o\lambda o.o\lambda o.o\lambda o.o$

  Endkonstellation: -

  gesperrte Farben: -

  Beschreibung: Viertes Tutoriallevel, das das System der Schrittanzahllevel erklärt.

- Level 7

  Typ: Multiple-Choice

  Anfangskonstellation: $(\lambda x.x)y$

  Wahlmöglichkeiten:

  1. y
  2. $\lambda x.x$
  3. yy

  Endkonstellation: y

  Beschreibung: Fünftes Tutoriallevel, in dem das System der Multiple-Choice Levels erklärt wird.

- Level 8

  Typ: Multiple-Choice

  Anfangskonstellation: $(\lambda x.\lambda y.x)z$

  Wahlmöglichkeiten:

  1. $\lambda y.z$
  2. $\lambda x.x$

3. $\lambda x.z$

Endkonstellation: $\lambda y.z$

- Level 9

  Typ: Einfärbelevel mit Schrittanzahl

  Schrittanzahl: 10

  Anfangskonstellation: $\lambda o.oo\ \lambda o.oo$

  Endkonstellation: -

  gesperrte Farben: -

- Level 10

  Typ: Multiple-Choice

  Anfangskonstellation: $(\lambda x.\lambda y.xx)uv$

  Wahlmöglichkeiten:

  1. uu
  2. vv
  3. $\lambda y.u$

  Endkonstellation: λuu

- Level 11

  Typ: Einfügelevel

  Anfangskonstellation: yx

  Endkonstellation: yyx

  gesperrte Farben: y, x

- Level 12

  Typ: Multiple-Choice

  Anfangskonstellation: $(\lambda x.x((\lambda y.y)(\lambda z.z))$

  Wahlmöglichkeiten:

  1. $\lambda y.y$
  2. z
  3. $\lambda z.z$

  Endkonstellation: $\lambda z.z$

6 Java Klassendokumentation

Zugunsten der Internationalität und somit für die bessere Wiederverwendbarkeit als Dokumentation für den Quellcode, sind die Beschreibungen der folgenden Klassen in englischer Sprache verfasst.

6.1 Package `de.croggle`

6.1.1 `public class AlligatorApp extends com.badlogic.gdx.Game`

Description

The central unit controlling the game. Manages the application lifecycle and is responsible for managing screens as well as the minor controllers.

Constructors

- `public AlligatorApp(Context context)`
  Creates the game using the given context and initializes all controllers and screens.

  Parameters

`context`	the Android Activity's context

Methods

- `public void create()`
  Is called by the application lifecycle on creation. Does all the initialization that hasn't been done by the constructor.
- `public void dispose()`
  Is called by the application lifecycle when the game is shut down. Should dispose everything that was allocated.
- `public AchievementController getAchievementController()`
  Returns the achievement controller which holds the information about achievements associated with the current profile.

Returns

the achievement controller

- `public AssetManager` **`getAssetManager`**`()`
  Returns the asset manager which controls all kinds of game media, e.g. graphics.

  Returns

  the asset manager

- `public Context` **`getContext`**`()`
  Returns the Android Context the game operates in.

  Returns

  the Android Context

- `public LevelPackagesController` **`getLevelPackagesController`**`()`
  Returns the level packages controller for level management purposes.

  Returns

  the level packages controller

- `public LocalizationManager` **`getLocalizationManager`**`()`
  Returns the localization manager which is used for translating strings to the appropriate language.

  Returns

  the localization manager

- `public PersistenceManager` **`getPersistenceManager`**`()`
  Returns the persistence manager which is responsible for all database operations.

  Returns

  the persistence manager

- `public ProfileController` **`getProfileController`**`()`
  Returns the profile controller which controls the information about the currently active profile.

  Returns

  the profile controller

- `public SettingController` **`getSettingController`**`()`
  Returns the setting controller that holds all profile-specific settings.

  Returns

  the setting controller

- `public StatisticController` **`getStatisticController()`**
  Returns the statistic controller which contains all information about the statistics of the active profile.

  Returns

  the statistic controller

- `public void` **`pause()`**
  Is called by the application lifecycle when the game is paused. Should save everything that has not been saved yet - such as the level progress - in case the game is shut down.

- `public void` **`render()`**
  Is called by the application lifecycle repeatedly and should update the game logic, as well as redraw the user interface.

- `public void` **`resize(int width, int height)`**
  Is called by the application lifecycle on resize.

  Parameters

`width`	the width that the screen will have afterwards.
`height`	the height that the screen will have afterwards.

- `public void` **`resume()`**
  Is called by the application lifecycle when the game returns from the pause state. Should rebuild the game the way it was before pausing (as far as possible).

6.1.2 `public class MainActivity extends AndroidApplication`

Description

Android backend that initializes the central ApplicationListener.

Constructors

- `public` **`MainActivity()`**

Methods

- `void` **`onCreate(Bundle savedInstance)`**
  Initializes the central ApplicationListener. Is called by the android lifecycle as soon as the app is started. On return, the inner app lifecycle of ApplicationListener is started.

6.2 Package `de.croggle.data`

6.2.1 `public class AssetManager extends com.badlogic.gdx.assets.AssetManager`

Description

This class is responsible for managing the different kinds of assets the apps needs to work flawlessly.

Constructors

- `public` **`AssetManager()`**

Methods

- `public Animation` **`getAnimation`**`(String identifier)`
  Loads the animation specified by the identifier.

  Parameters

  `identifier` a path to the requested animation, resolvable by the asset manager

  Returns

  the animation denoted by the given identifier

- `public BitmapFont` **`getFont`**`(String identifier)`
  Loads the font specified by the identifier.

  Parameters

  `identifier` a path to the requested bitmap font, resolvable by the asset manager

  Returns

  the bitmap font denoted by the given identifier

- `public Music` **`getMusic`**`(String identifier)`
  Loads the music specified by the identifier.

  Parameters

  `identifier` a path to the requested music, resolvable by the asset manager

Returns

the music denoted by the given identifier

- `public Sound **getSound**(String identifier)`
  Loads the sound specified by the identifier.

  Parameters

  `identifier` a path to the requested sound, resolvable by the asset manager

  Returns

  the sound denoted by the given identifier

- `public Texture **getTexture**(String identifier)`
  Loads the texture specified by the identifier.

  Parameters

  `identifier` a path to the requested texture, resolvable by the asset manager

  Returns

  the texture denoted by the given identifier

6.2.2 `public class LocalizationManager`

Description

Manager whose task it is to handle the translation of the app from one language to another. The strings should be defined in the usual Android strings xml file.

Constructors

- `public **LocalizationManager**(Context context)`
  Creates the LocalizationManager that uses the given Android context to achieve defined strings.

  Parameters

  `context` the context of the Android application

Methods

- `public String` **`getString`**`(int identifier)`
  Loads and returns the string identified by the given id. The strings should be defined in the common Android language xml files (values/strings). The language is decided by the system. The identifier can be accessed by `R.string.identifier`.

  Parameters

`indentifier`	the identifying number

6.3 Package `de.croggle.data.persistence`

6.3.1 `public class LevelProgress`

Description

Represents the progress saved by a user during one level in the database.

Constructors

- `public LevelProgress(long levelId, boolean solved, String currentBoard, int usedResets, int usedHints, int usedTime)`
  Constructs a new LevelProgress based on it's properties.

  Parameters

`profileId`	the id of the user's profile
`levelId`	the id of the level
`solved`	whether the level has been solved
`currentBoard`	the serialized representation of the current board
`usedResets`	the number of resets used by the user
`usedHints`	the number of hints used by the user
`usedTime`	the time spent in the level by the user

- `public LevelProgress(Cursor cursor)`
  Constructs a new LevelProgress using a cursor to the correct database row.

  Parameters

`cursor`	the cursor

Methods

- `public String getCurrentBoard()`
  Gets the serialized version of the current board.

  Returns

  the currently used board

- `public long getLevelId()`
  Gets the id of the level.

  Returns

  the level id

- `public int` **`getUsedHints`**`()`
  Gets the number of hints used by the user.

  Returns

  the number of times the user used hints

- `public int` **`getUsedResets`**`()`
  Gets the number of resets triggered by the user.

  Returns

  the number of times the user resetted the level

- `public int` **`getUsedTime`**`()`
  Gets the time spent by the user in the level.

  Returns

  the time in seconds

- `public boolean` **`isSolved`**`()`
  Gets whether the level has been solved.

  Returns

  true if the level has been solved, false otherwise

- `public void` **`setCurrentBoard`**`(String currentBoard)`
  Sets the serialized version of the current board.

  Parameters

  `currentBoard` the currently used board

- `public void` **`setLevelId`**`(long levelId)`
  Sets the id of the level.

  Parameters

  `levelId` the level id

- `public void` **`setSolved`**`(boolean solved)`
  Sets whether the level has been solved.

  Parameters

  `solved` true if the level has been solved, false otherwise

- `public void` **`setUsedHints`**`(int usedHints)`
  Sets the number of hints used by the user.

Parameters

`usedHints`	the number of times the user used hints

- `public void` **`setUsedResets`**`(int usedResets)`
  Sets the number of resets by the user.

  Parameters

`usedResets`	the number of times the user reseted the level

- `public void` **`setUsedTime`**`(int usedTime)`
  Sets the time spent by the user in the level.

  Parameters

`usedTime`	the time in seconds

6.3.2 `public class Setting`

Description

Represents the settings of a certain profile in the database.

Constructors

- `public` **`Setting`**`(float volumeMusic, float volumeEffect, boolean zoomEnabled, boolean colorblindEnabled)`
  Constructs a new LevelProgress based on its properties.

  Parameters

`volumeMusic`	the volume of the music.
`volumeEffect`	the volume of the effects.
`zoomEnabled`	determines whether the zoom button is enabled or not.
`colorblindEnabled`	determines whether colorblind mode is enabled or not.

- `public` **`Setting`**`()`
  Creates a new default setting.

Methods

- `public float` **`getVolumeEffects`**`()`
  Gets the effect volume.

  Returns

  the volume of the effects

- `public float` **`getVolumeMusic`**`()`
  Gets the music volume.

  Returns

  the volume of the music

- `public boolean` **`isColorblindEnabled`**`()`
  Gets whether the colorblind mode is enabled or disabled.

  Returns

  true if the colorblind mode is enabled, false otherwise

- `public boolean` **`isZoomEnabled`**`()`
  Gets whether zoom button is enabled or disabled.

  Returns

  true if the zoom button is enabled, false otherwise

- `public void` **`setColorblindEnabled`**`(boolean colorblindEnabled)`
  Sets the colorblind mode to enabled or disabled.

  Parameters

`colorblindEnabled`	true for enabling colorblind mode, false for disabling it

- `public void` **`setVolumeEffects`**`(float volumeEffects)`
  Sets the effect volume.

  Parameters

`volumeEffects`	the volume of the effects

- `public void` **`setVolumeMusic`**`(float volumeMusic)`
  Sets the music's volume.

  Parameters

`volumeMusic`	the volume of the music

- `public void` **`setZoomEnabled`**`(boolean zoomEnabled)`
  Sets zoom button to enabled or disabled.

  Parameters

`zoomEnabled`	true for enabling zoom, false for disabling it

6.3.3 `public class SettingController`

Description

Controller which handles the different settings currently applied.

Constructors

- `public` **`SettingController`**`(AlligatorApp game)`
  Creates a new SettingController. On initialization the active setting is set to null.

  Parameters

`game`	the backreference to the central game object

Methods

- `public void` **`changeCurrentSetting`**`(String profileName)`
  Loads the setting which belongs to the user identified with the profile name and sets it as the current setting.

  Parameters

`profileName`	the name of the user whose settings are loaded

  Throws

`IllegalArgumentException`	whenever the string does not represent a profile in the database

- `public void` **`editCurrentSetting`**`(Setting newSetting)`
  Replaces the current setting with a new one. The new setting gets stored in the database and overwrites the values of the old setting.

  Parameters

`newSetting`	the setting used to replace the currently active setting

- `public Setting` **`getCurrentSetting`**`()`
  Returns the current setting.

  Returns

  the currently active settings

6.3.4 `public class Statistic`

Description

Represents - in the database - everything there is to know about the things a user has done within the game .

Constructors

- `public` **`Statistic()`**
  Creates a new default statistic.

Methods

- `public int` **`getAlligatorsEaten()`**
  Gets the total number of alligators eaten during beta reductions.

  Returns

  the number of eaten alligators

- `public int` **`getAlligatorsPlaced()`**
  Gets the total number of alligators placed in the placement mode.

  Returns

  the number of placed alligators

- `public int` **`getEggsHatched()`**
  Gets the total number of eggs hatched during beta reductions.

  Returns

  the number of hatched eggs

- `public int` **`getEggsPlaced()`**
  Gets the total number of eggs placed in the placement mode.

  Returns

  the number of placed eggs

- `public int` **`getLevelsComplete()`**
  Gets the number of completed levels.

  Returns

  the number of completed levels

- `public int` **`getPackagesComplete()`**
  Gets the number of completed packages.

Returns

the number of completed packages by the player

- `public int` **`getPlaytime`**`()`
  Gets the playtime.

  Returns

  the time spent playing

- `public int` **`getRecolorings`**`()`
  Gets the number of recoloring actions.

  Returns

  the number of recoloring actions

- `public int` **`getResetsUsed`**`()`
  Gets the number of resets used.

  Returns

  the number of resets used by the player

- `public int` **`getUsedHints`**`()`
  Gets the number of hints used.

  Returns

  the number of hints used

- `public void` **`setAlligatorsEaten`**`(int alligatorsEaten)`
  Sets the number of eaten alligators.

  Parameters

  `alligatorsEaten` the new number of eaten alligators

- `public void` **`setAlligatorsPlaced`**`(int alligatorsPlaced)`
  Sets the number of placed alligators.

  Parameters

  `alligatorsPlaced` the new number of placed alligators

- `public void` **`setEggsHatched`**`(int eggsHatched)`
  Sets the number of hatched eggs.

  Parameters

  `eggsHatched` the new number of hatched eggs

- `public void` **`setEggsPlaced`**`(int eggsPlaced)`
  Sets the number of placed eggs.

  Parameters

  `eggsPlaced` the new number of placed eggs

- `public void` **`setLevelsComplete`**`(int levelsComplete)`
  Sets the number of completed levels.

  Parameters

  `levelsComplete` the new number of completed levels

- `public void` **`setPackagesComplete`**`(int packagesComplete)`
  Sets the number of completed packages.

  Parameters

  `packagesComplete` the new number of completed packages

- `public void` **`setPlaytime`**`(int playtime)`
  Sets the playtime.

  Parameters

  `playtime` the new playtime

- `public void` **`setRecolorings`**`(int recolorings)`
  Sets the number of recoloring actions.

  Parameters

  `recolorings` the new number of recoloring actions

- `public void` **`setResetsUsed`**`(int resetsUsed)`
  Sets the number of resets used.

  Parameters

  `resetsUsed` the new number of resets used

- `public void` **`setUsedHints`**`(int usedHints)`
  Sets the number of hints used.

  Parameters

  `usedHints` the new number of hints used

6.3.5 `public class StatisticController implements de.croggle.data.persistence.StatisticsDeltaProcessor`

Description

Controller that holds and controls the active Statistic. The active Statistic is the one that belongs to the active profile.

Constructors

- `public` **`StatisticController`**`(AlligatorApp game)`
  Creates a new controller. On initialization the active statistic is set to null.

  Parameters

`game`	the backreference to the central game object

- `public` **`StatisticController`**`(Statistic statistic, AlligatorApp game)`
  Creates a controller with the given statistic as initial active statistic.

  Parameters

`statistic`	the statistic to set as active
`game`	the backreference to the central game object

Methods

- `public void` **`changeCurrentStatistic`**`(String profileName)`
  Loads the statistic which belongs to the user identified by profile name and sets it as the currently active statistic.

  Parameters

`profileName`	the name of the user whose statistic is loaded

  Throws

`IllegalArgumentException`	whenever the string does not represent a profile in the database

- `public void` **`editCurrentStatistic`**`(Statistic newStatistic)`
  Replaces the current statistic with a new one. The new statistic is stored in the database and overwrites the values of the old statistic.

  Parameters

`newStatistic`	the statistic used to replace the currently active statistic

- `public Statistic` **`getCurrentStatistic`**`()`
  Returns the active statistic that belongs to the active profile.

Returns

the active statistic

- `public Statistic` **`getStatistic`**`(String profileName)`
  Returns the statistic of the profile that is identified by the given string.

  Parameters

`profileName`	the identifier of the profile whose statistic should be loaded

  Returns

  the statistic of the specified profile

  Throws

`IllegalArgumentException`	whenever the string does not represent a profile in the database

- `public void` **`processDelta`**`(Statistic statisticsDelta)`
  Adds the delta to the values of the active statistic. There needs to be an active statistic that is not null.

6.3.6 `public interface interface StatisticsDeltaProcessor`

Description

A listener that is to be implemented by all classes that need to process data from the statistic values that occur during a level.

Methods

- `public void` **`processDelta`**`(Statistic statisticsDelta)`
  Evaluates and processes the statistic changes that occurred during a level, e.g. updates the database. The changes are packed as a Statistic object. The statistics-Delta is changed afterwards, so no references to it should be held.

  Parameters

`statisticsDelta`	the packed statistic changes

6.4 Package `de.croggle.data.persistence.manager`

6.4.1 `public class AchievementManager extends de.croggle.data.persistence.manager.TableManager`

Description

A concrete table manager which is responsible for managing the SQLite table that stores the unlocked achievements of the different profiles.

Attributes

- `static final String` **`CREATE_TABLE`**
  The string used for creating the achievement table via a sql query.
- `static final String` **`KEY_ACHIEVEMENT_ID`**
  Name of the column that stores the achievement IDs.
- `static final String` **`KEY_ACHIEVEMENT_State`**
  Name of the column that stores the achievement states.
- `static final String` **`KEY_PROFILE_NAME`**
  Name of the column that stores the profile names. Those names are used as the primary key.
- `static final String` **`TABLE_NAME`**
  The name of the table.

Constructors

- **`AchievementManager`**`(Context context)`
  Creates a new AchievementManager used for managing the achievement table.

  Parameters

`context`	the context used for accessing the database

Methods

- `void` **`addUnlockedAchievement`**`(String profileName, Achievement achievement)`
  Adds a new unlocked achievement to the table.

  Parameters

`profileName`	the name of the profile to which the unlocked achievement belongs
`achievement`	contains the values to be stored in the table

- `void` **`deleteUnlockedAchievements`**`(String profileName)`
  Deletes all achievements that were unlocked by the user with the given profile name from the table.

  Parameters

  `profileName` the name of the profile whose unlocked achievements are deleted

- `List<Achievement>` **`getUnlockedAchievements`**`(String profileName)`
  Returns all achievements stored in the table that were unlocked by the user with the given profile name.

  Parameters

  `profileName` the name of the user whose unlocked achievements are searched for

  Returns

  a list of all achievements unlocked by the user

6.4.2 `public class DatabaseHelper extends android.database.sqlite.SQLiteOpenHelper`

Description

This class is responsible for creating and managing the database with its different tables.

Constructors

- `public` **`DatabaseHelper`**`(Context context)`
  Creates a new DatabaseHelper which is used for managing the database.

  Parameters

  `context` the context used to create the database

Methods

- `public void` **`onCreate`**`(SQLiteDatabase db)`
  Creates all tables if they don't already exist.

- `public void` **`onUpgrade`**`(SQLiteDatabase db, int oldVersion, int newVersion)`
  Deletes the old database and creates a new one.

6.4.3 `public class LevelProgressManager extends de.croggle.data.persistence.manager.TableManager`

Description

A concrete table manager which is responsible for managing the SQLite table that stores the level progresses of the different profiles.

Attributes

- `static final String` **`CREATE_TABLE`**
  The string used for creating the level progress table via a sql query.
- `static final String` **`KEY_CURRENT_BOARD`**
  Name of the column that stores the current board.
- `static final String` **`KEY_LEVEL_ID`**
  Name of the column that stores the level id. The IDs are used as the secondary key.
- `static final String` **`KEY_PROFILE_NAME`**
  Name of the column that stores the profile names. The names are used as the primary key.
- `static final String` **`KEY_SOLVED`**
  Name of the column that stores whether the level has been solved or not.
- `static final String` **`KEY_USED_HINTS`**
  Name of the column that stores the number of used hints.
- `static final String` **`KEY_USED_RESETS`**
  Name for the column that stores the number of used resets.
- `static final String` **`KEY_USED_TIME`**
  Name of the column that stores the amount of used time.
- `static final String` **`TABLE_NAME`**
  The name of the table.

Constructors

- **`LevelProgressManager`**`(Context context)`
  Creates a new LevelProgressManager which manages the level progress table.

  Parameters

  `context` the context used for accessing the database

Methods

- `void` **`addLevelProgress`**`(String profileName, LevelProgress levelProgress)`
  Adds a new level progress to the table.

  Parameters

`profileName`	the name of the profile to which the level progress belongs
`levelProgress`	the level progress contains the values to be stored in the table

- `void` **`deleteLevelProgresses`**`(String profileName)`
  Deletes all level progresses which belong to the profile identified by the profile name from the table.

  Parameters

`profileName`	the name of the profile to which the level progresses belong

- `LevelProgress` **`getLevelProgress`**`(String profileName, long levelId)`
  Searches the table for a level progress that belongs to the profile identified by the profile name and whose level ID matches the level ID stored in level progress.

  Parameters

`profileName`	the name of the profile to which the level progresses belong
`levelId`	the level ID of the searched-for level progress

  Returns

  the found level progress, null if no level progress is found

- `void` **`updateLevelProgress`**`(String profileName, LevelProgress levelProgress)`
  Searches the table for a level progress that belongs to the profile identified by the profile name and whose level ID matches the level ID stored in levelProgress. The values of the found level progress are overwritten by the new level progress.

  Parameters

`profileName`	the name of the profile to which the level progresses belong
`levelProgress`	the level progress whose values are used for overwriting the old level progress

6.4.4 `public class PersistenceManager`

Description

This class provides methods for storing and loading profile-specific data.

Constructors

- `public` **`PersistenceManager`**`(AlligatorApp game)`
  Creates a new PersistenceManager and initializes the different managers.

  Parameters

`game`	the backwards reference to the central game object

Methods

- `public void` **`addProfile`**`(Profile profile)`
  Stores a new profile with the default settings and statistics.

  Parameters

`profile`	the profile to be stored

- `public void` **`deleteProfile`**`(String profileName)`
  Deletes the profile with the given name (all entries referenced by it are also deleted).

  Parameters

`profileName`	the name of the profile to be deleted

- `public void` **`editProfile`**`(String profileName, Profile profile)`
  Overwrites the profile identified by the given name with the values of the new profile. Every reference to the profile name is updated.

  Parameters

`profileName`	the string to identify the profile which is to be edited
`profile`	contains the values used for overwriting the old profile

- `public void` **`editSetting`**`(String profileName, Setting newSetting)`
  Overwrites the setting of the profile identified by the given name with the values of the new setting.

  Parameters

`profileName`	the name of the profile to which the setting belongs
`newSetting`	contains the new values used for overwriting the old setting

- `public void` **`editStatistic`**`(String profileName, Statistic newStatistic)`
  Overwrites the statistic of a specific profile identified by the given profile name with the new statistic.

Parameters

`profileName`	the name of the profile to which the statistic belongs
`newStatistic`	contains the new values used for overwriting the old statistic

- `public List<Profile>` **`getAllProfiles`**`()`
  Returns all stored profiles.

  Returns

  a list of all stored profiles

- `public List<Achievement>` **`getAllUnlockedAchievements`**`(String profileName)`
  Returns all achievements unlocked by the user identified by the given profile name.

  Parameters

`profileName`	the name of the profile whose unlocked achievements are searched for

  Returns

  a list containing all achievements unlocked by the user

- `public LevelProgress` **`getLevelProgress`**`(String profileName, int levelID)`
  Returns the level progress whose level ID matches the given level id and which belongs to the profile with the given profile name.

  Parameters

`profileName`	the name of the profile to which the levelProgress belongs
`levelID`	the level ID of the level progress

  Returns

  the found level progress, null if no level progress is found

- `public Setting` **`getSetting`**`(String profileName)`
  Returns the setting of the profile with the given profile name.

  Parameters

`profileName`	the name of the profile to which the setting belongs

  Returns

  the found setting, null if no setting is found

- `public Statistic` **`getStatistic`**`(String profileName)`
  Returns the statistic of the profile with the givne name.

Parameters

`profileName`	the name of the profile to which the statistic belongs

Returns

the found statistic, null if no statistic is found

- `public Profile` **`loadProfile`**`(String profileName)`
  Returns the profile with the given profile name.

  Parameters

`profileName`	the name of the profile which is to be loaded

  Returns

  the profile which has been loaded, null if there is no profile with this name

- `public void` **`saveLevelProgress`**`(String profileName, LevelProgress levelProgress)`
  Saves a level progress for a specific profile identified by the given profile name. If there already is an entry for the profile which has the same level id as the level id of the level progress, the old entry gets overwritten.

  Parameters

`profileName`	the name of the profile to which the statistic belongs
`levelProgress`	contains the new values used for storing the level progress or overwrite the old level progress

- `public void` **`saveUnlockedAchievement`**`(String profileName, Achievement achievement)`
  Saves an unlocked achievement for a specific profile identified by the given profile name.

  Parameters

`profileName`	the name of the user that unlocked the achievement
`achievement`	contains the values to be stored

6.4.5 `public class ProfileManager extends de.croggle.data.persistence.manager.TableManager`

Description

A concrete table manager which is responsible for managing the SQLite table that stores the different profiles.

Attributes

- `static final String` **`CREATE_TABLE`**
  The string used for creating the profile table via a sql query.
- `static final String` **`KEY_PICTUREPATH`**
  Name of the column that stores the path to the profile pictures.
- `static final String` **`KEY_PROFILE_NAME`**
  Name of the column that stores the profile names. The names are used as the primary key.
- `static final String` **`TABLE_NAME`**
  The name of the table.

Constructors

- **`ProfileManager`**`(Context context)`
  Creates a new ProfileManager which manages the profile table.

  Parameters

  `context` the context used for accessing the database

Methods

- `void` **`addProfile`**`(Profile profile)`
  Adds a new profile to the table.

  Parameters

  `profile` contains the values to be stored in the table

- `void` **`deleteProfile`**`(String profileName)`
  Deletes the profile whose name matches the given profile name from the table.

  Parameters

  `profileName` the name of the user whose profile is to be deleted

- `void editProfile(String profileName, Profile profile)`
  Searches the table for a profile whose name matches the given profile name and overwrites its values with the values of the new profile.

  Parameters

`profileName`	the name of the profile which is edited.
`profile`	contains the values used for overwriting the old entry

- `List<Profile> getAllProfiles()`
  Returns all profiles stored in the table.

  Returns

  the list of all profiles

- `Profile getProfile(String profileName)`
  Searches the table for a profile whose name matches the given profile name.

  Parameters

`profileName`	the name of the searched profile

  Returns

  the found profile, null if no profile is found

6.4.6 `public class SettingManager extends de.croggle.data.persistence.manager.TableManager`

Description

A concrete table manager which is responsible for managing the SQLite table that stores the settings of the different profiles.

Attributes

- `static final String CREATE_TABLE`
  The string used for creating the setting table via a sql query.
- `static final String KEY_COLORBLIND_ENABLED`
  Name of the column that stores the information whether the colorblind mode is enabled or not.
- `static final String KEY_PROFILE_NAME`
  Name of the column that stores the profile names. The names are used as the primary key.

- `static final String` **`KEY_VOLUME_EFFECTS`**
  Name of the column that stores the volume of the effects.

- `static final String` **`KEY_VOLUME_MUSIC`**
  Name of the column that stores the volume of the music.

- `static final String` **`KEY_ZOOM_ENABLED`**
  Name of the column that stores the information whether zoom is enabled or not.

- `static final String` **`TABLE_NAME`**
  The name of the table.

Constructors

- **`SettingManager`**`(Context context)`
  Creates a new SettingManager which manages the setting table.

  Parameters

`context`	used for accessing the database

Methods

- `void` **`addSetting`**`(String profileName, Setting setting)`
  Adds a new setting to the table.

  Parameters

`profileName`	the name of the profile whose setting is added to the table
`setting`	contains the values to be stored in the table

- `void` **`deleteSetting`**`(String profileName)`
  Deletes the setting which belongs to the profile identified by the given profile name from the table.

  Parameters

`profileName`	the name of the profile whose setting is to be deleted

- `void` **`editSetting`**`(String profileName, Setting setting)`
  Searches the table for a setting which belongs to the profile identified by the given profile name and overwrites its values with the values of the new setting.

  Parameters

`profileName`	the name of the profile whose setting is edited
`setting`	the setting whose values are used for overwriting the old setting

- `Setting` **`getSetting`**`(String profileName)`
  Searches the table for a setting which belongs to the profile identified by the given profile name.

  Parameters

  `profileName` the name of the profile whose setting is searched for

  Returns

  the found setting, null if no setting is found

6.4.7 `public class StatisticManager extends de.croggle.data.persistence.manager.TableManager`

Description

A concrete table manager is responsible for managing the SQLite table that stores the statistics of the different profiles.

Attributes

- `static final String` **`CREATE_TABLE`**
  The string used for creating the statistic table via a sql query.
- `static final String` **`KEY_ALLIGATORS_EATEN`**
  Name of the column that stores the number of eaten alligators.
- `static final String` **`KEY_ALLIGATORS_PLACED`**
  Name of the column that stores the number of placed alligators.
- `static final String` **`KEY_EGGS_HATCHED`**
  Name of the column that stores the number of hatched eggs.
- `static final String` **`KEY_EGGS_PLACED`**
  Name of the column that stores the number of placed eggs.
- `static final String` **`KEY_LEVELS_COMPLETE`**
  Name of the column that stores the number of completed levels.
- `static final String` **`KEY_PACKAGES_COMPLETE`**
  Name of the column that stores the number of completed packages.
- `static final String` **`KEY_PLAYTIME`**
  Name of the column that stores the playtimes.

- `static final String` **`KEY_PROFILE_NAME`**
  Name of the column that stores the profile names. The names are used as the primary key.
- `static final String` **`KEY_RECOLORINGS`**
  Name of the column that stores the number of recoloring actions.
- `static final String` **`KEY_USED_HINTS`**
  Name of the column that stores the number of used hints.
- `static final String` **`KEY_USED_RESETS`**
  Name of the column that stores the number of used resets.
- `static final String` **`TABLE_NAME`**
  The name of the table.

Constructors

- **`StatisticManager`**`(Context context)`
  Creates a new StatisticManager which manages the statistic table.

  Parameters

`context`	used for accessing the database

Methods

- `void` **`addStatistic`**`(String profileName, Statistic statistic)`
  Adds a new statistic to the table.

  Parameters

`profileName`	the name of the profile whose statistic is added to the table
`statistic`	contains the values to be stored in the table

- `void` **`deleteStatistics`**`(String profileName)`
  Deletes the statistic which belongs to the profile identified by the given profile name from the table.

  Parameters

`profileName`	the name of the profile whose statistic is deleted

- `void` **`editStatistic`**`(String profileName, Statistic statistic)`
  Searches the table for a statistic which belongs to the profile identified by the given profile name and overwrites its values with the values of the new statistic.

  Parameters

`profileName`	the name of the profile whose statistic is edited
`statistic`	the statistic whose values are used for overwriting the old statistic

- `Statistic` **`getStatistic`**`(String profileName)`
  Searches the table for a statistic which belongs to the profile identified by the given profile name.

  Parameters

`profileName`	the name of the profile whose statistic is loaded

  Returns

  the found statistic, null if no statistic is found

6.4.8 `public abstract class TableManager`

Description

An abstract superclass for all classes which manage tables.

Attributes

- `protected SQLiteDatabase` **`database`**
  The database in which the table is stored.
- `protected DatabaseHelper` **`databaseHelper`**
  The DatabaseHelper is used for accessing the database in which the table is stored.

Constructors

- **`TableManager`**`(Context context)`
  Creates a new TableManager, which manages a specific table from the database that belongs to the given context.

  Parameters

`context`	the context that is used for opening or, if needed, creating the database

Methods

- `void` **`close`**`()`
  Closes the open table. Must be called at the end of reading and writing operations.

  Throws

`SQLException`	the exception is thrown if the database could not be accessed

- `void` **`open()`**
  Prepares the table for read and write operations. Must be called before every access to the table.

  Throws

`SQLException`	the exception is thrown if the database could not be accessed

6.5 Package `de.croggle.game`

6.5.1 `public class Color`

Description

A color represents a variable name.

Constructors

- `public Color(int id)`
  Creates a color with the given id. The id needs to be between 0 and 29 and represents a certain "real" color according to the ColorController.

  Parameters

  `id` the identifying color id

  Throws

  `IllegalArgumentException` when the id is not a number between 0 and 29

Methods

- `public int getId()`
  Gets the globally unique color id between 0 and 29.

6.5.2 `public class ColorController`

Description

The color controller manages colors in the game. It is mainly responsible for mapping the virtual colors of board objects consistently on real life colors. Additionally, it provides functionality for generating new colors used on boards if needed by the simulator after applying recolor rules.
The terms "blocked" and "usable" for colors refer to which colors are blocked by the level specification (blocked) and which are used by the game to let the user recolor elements (usable).

Constructors

- `public ColorController()`
  Initializes the color controller with no colors blocked, no colors usable and no colors in use.

Methods

- `public void` **`addBlockedColor`**`(Color color)`
  Adds a model color to the list of colors that may not be used to recolor board objects.

  Parameters

  `color` a color to be blocked

- `public void` **`addUsableColor`**`(Color color)`
  Adds a model color to the list of the colors usable by the user to recolor board objects.

  Parameters

  `color` a color to be marked as usable

- `public Color` **`getAssociatedColor`**`(Color color)`
  Performs a lookup upon a given libgdx.Color and returns the BoardObject Color represented by it.

  Parameters

  `color` the color which represents the color to be looked up

  Returns

  a model color that is represented by the given libgdx Color

- `public Color` **`getRepresantation`**`(Color color)`
  Performs a lookup upon a given model. Color to consistently map it to a real life color.

  Parameters

  `color` the color whose real life color is to be looked up

  Returns

  an libgdx color to be actually rendered to represent the virtual color of a Board-Object

- `public Color[]` **`getUsableColors`**`()`

  Returns

  an array of all currently usable colors

- `public boolean` **`isBlocked`**`(Color color)`
  Look up whether a given color is blocked, i.e. it may not occur in the list of usable colors

  Parameters

  `color`

  Returns

  whether the given color is blocked or not

- `public boolean` **`isUsable`**`(Color color)`

  Parameters

  `color` the color whose usability should be tested

  Returns

  whether the given color is usable or not

- `public Color` **`requestColor`**`()`
  Returns a new color to be used by the simulator on a board and assigns an actual libgdx Color to it. This is equivalent to calling `requestColor(allUsedColors)`, with `allUsedColors` being an array of all colors used on the board.

  Returns

  a new color to be used on the board

  Throws

  `ColorOverflowException` if there is no color available

- `public Color` **`requestColor`**`(Color[] usedColors)`
  Returns a color which does not appear in `usedColors` to be used by the simulator on a board for recoloring. If all available colors are in usedColors, a new color is created.

  Parameters

  `usedColors` a set of colors which are already used

  Returns

  a color to be used on the board

  Throws

  `ColorOverflowException` if there is no color available

6.5.3 `public class ColorOverflowException extends java.lang.Exception`

Description

The exception is thrown whenever a lambda term contains more than 30 different colors. This mostly happens during alpha conversion.

Constructors

- `public` **`ColorOverflowException`**`()`
  Creates a new instance of the exception with the default constructor.
- `public` **`ColorOverflowException`**`(String message)`
  Creates a new instance of the exception with the given error message.

  Parameters

  `message` a message describing the cause of the exception that occured

6.5.4 `public class GameController implements de.croggle.game.event.BoardEventListener`

Description

Central controller within which the actual playing of the level is controlled. Additionally, it handles the consequences of finishing a level and distributes the changes.

Constructors

- `public` **`GameController`**`(Level level)`
  Creates a new game controller for the given level.

  Parameters

  `level` the level the GameController should work with

Methods

- `public void` **`onAgedAlligatorVanishes`**`(AgedAlligator alligator)`
  Would register this in the statisticsDelta, but currently there is no value like this in the statistic.
- `public void` **`onBoardRebuilt`**`(Board board)`
  Resets all necessary values of the statisticsDelta.

- `public void` **`onCompletedLevel`**`()`
  Called when the level is completed. Writes the important results into the database and eventually tells the achievement controller which achievements were achieved. Passes the statisticsDelta to all of its listeners.

- `public void` **`onEat`**`(ColoredAlligator eater, InternalBoardObject eatenFamily)`
  Registers the amount of alligators and eggs eaten in the statisticsDelta.

- `public void` **`onObjectRecolored`**`(InternalBoardObject recoloredObject)`
  Registers the recoloring of an object in the statisticsDelta.

- `public void` **`onReplace`**`(Egg replacedEgg, InternalBoardObject bornFamily)`
  Registers the hatched egg and the born family in the statisticsDelta.

- `public void` **`register`**`(StatisticsDeltaProcessor listener)`
  Registers a listener to whom the statisticsDelta should be passed after the completion of the level.

  Parameters

`listener`	the listener

- `public void` **`registerBoardEventListener`**`(BoardEventListener listener)`
  Registers a listener to which board events should be sent.

  Parameters

`listener`	the listener which should receive the events

- `public void` **`unregister`**`(StatisticsDeltaProcessor listener)`
  Unregisters the statistic listener.

  Parameters

`listener`	the listener

- `public void` **`unregisterBoardEventListener`**`(BoardEventListener listener)`
  Unregisters a board event listener so that it won't receive future events.

  Parameters

`listener`	the listener to unregister

6.5.5 `public class Simulator`

Description

The Simulator is the instance which evaluates the Board given to it. It can also undo steps done in the evaluation process.

Constructors

- `public **Simulator**(Board entranceBoard, ColorController colorController, BoardEventMessenger boardMessenger)`
  Creates a new Simulator.

  Parameters

`entranceBoard`	the board that is evaluated by this simulator
`colorController`	the color controller used for recoloring during evaluation
`boardMessenger`	the board messenger used for sending events during evaluation

  Throws

`IllegalBoardException`	if the `entranceBoard` is not a valid board

Methods

- `public Board **evaluate**(Board currentBoard)`
  Evaluates one step in the Lambda Calculus.

  Returns

  the board after said step

  Throws

`ColorOverflowException`	if recoloring occurs and there is no color available
`AlligatorOverflowException`	if there are more than the max. allowed amount of BoardObjects on the board after the evaluation step

- `public Board **reset**()`
  Reverses the board into the position it had upon entering simulation mode.

  Returns

  the board in said state

- `public Board **undo**()`
  Reverses the last evaluation step.

  Returns

  the board, in its status before the last evaluation step

6.6 Package `de.croggle.game.achievement`

6.6.1 `public abstract class Achievement`

Description

A reward given to the player for completing a special feat, e.g. playing for a certain amount of time or beating a certain amount of levels.

Constructors

- `public` **`Achievement()`**

Methods

- `public String` **`getDescription`**`(int index)`
  Returns a description that describes how to reach the achievement.

  Parameters

  `index` the stage which the description should describe

  Returns

  the text which is shown in order to describe the achievement

- `public String` **`getEmblemPath`**`(int index)`
  Returns the path to the picture that represents the achievement.

  Parameters

  `index` the stage index for which the emblem path should be returned

  Returns

  the path leading to the emblem of the achievement

- `public int` **`getId`**`()`
  Returns the id of the achievement that identifies it.

  Returns

  the achievement id

- `public abstract int` **`requirementsMet`**`()`
  Calculates the index of the stage the achievement has reached, according to the current statistics.

Returns

the updated index

6.6.2 `public class AchievementController implements de.croggle.data.persistence.StatisticsDeltaProcessor`

Description

Controller responsible for the achievements and for checking whether achievements have been achieved.

Constructors

- `public` **`AchievementController`**`(AlligatorApp game)`
  Creates a new Controller. On initialization the unlocked achievements are set to null.

  Parameters

`game`	the backreference to the central game object

Methods

- `public void` **`changeUnlockedAchievements`**`(String profileName)`
  Loads the achievements unlocked by the user with the name `profileName` and sets them as the unlocked achievements.

  Parameters

`profileName`	the name of the user which unlocked achievements are loaded

  Returns

  true if the change was successful, false otherwise

  Throws

`IllegalArgumentException`	whenever the string does not represent a profile in the database

- `public List<Achievement>` **`getAvailableAchievements`**`()`
  Get all achievements available in the game.

  Returns

  a list of available achievements

- `public List<Achievement>` **`getLatestUnlockedAchievements`**`()`
  Returns the list of achievements that were unlocked during the last completed level The list may be emptied afterwards, so no references should be held.

  Returns

  the list of latest unlocked achievements

- `public List<Achievement>` **`getUnlockedAchievements`**`()`
  Get the achievements unlocked by the currently active user.

  Returns

  a list of unlocked achievements

- `public void` **`initiateAvailableAchievements`**`()`
  Initiates the available achievements.

- `public void` **`processDelta`**`(Statistic statisticsDelta)`
  Checks whether the new statistic changes in a level cause new achievements to get unlocked. In this case it sets them as unlocked; the newly unlocked achievements can be accessed via the `getLatestUnlockedAchievements()` method.

  Parameters

`statisticsDelta`	the packed statistic changes

- `public List<Achievement>` **`processStatisticsDelta`**`(Statistic statisticsDelta)`
  Receives statistics delta from the just finished level and processes it.

  Parameters

`statisticsDelta`	changes within the statistic of an account which occured during the completion of a level

  Returns

  list of achieved achievements (might be empty)

6.6.3 `public class AlligatorsEatenAchievement extends de.croggle.game.achievement.Achievement`

Description

Achievements which are awarded for reaching certain, specified amounts of eaten alligators.

Constructors

- public **`AlligatorsEatenAchievement()`**

Methods

- public int **`requirementsMet()`**
  Calculates the index of the stage the achievement has reached, according to the current statistics.
  (*documentation inherited from* *`de.croggle.game.achievement.Achievement`*)

6.6.4 `public class AlligatorsEatenPerLevelAchievement extends de.croggle.game.achievement.PerLevelAchievement`

Description

An achievement the player gets for having at least a certain amount of alligators eaten in any level.

Constructors

- public **`AlligatorsEatenPerLevelAchievement()`**

Methods

- public int **`requirementsMet()`**
  Calculates the index of the stage the achievement has reached, according to the current statistics.
  (*documentation inherited from* *`de.croggle.game.achievement.Achievement`*)
  (*documentation inherited from* *`de.croggle.game.achievement.PerLevelAchievement`*)

6.6.5 `public class HintPerLevelAchievement extends de.croggle.game.achievement.PerLevelAchievement`

Description

An achievement the player gets for using no hints (or a certain number of them) in a level.

Constructors

- `public HintPerLevelAchievement()`

Methods

- `public int requirementsMet()`
  Calculates the index of the stage the achievement has reached, according to the current statistics.
  (*documentation inherited from `de.croggle.game.achievement.Achievement`*)
  (*documentation inherited from `de.croggle.game.achievement.PerLevelAchievement`*)

6.6.6 `public class LevelAchievement extends de.croggle.game.achievement.Achievement`

Description

An achievement that rewards completing a given amount of levels.

Constructors

- `public LevelAchievement()`

Methods

- `public int requirementsMet()`
  Calculates the index of the stage the achievement has reached, according to the current statistics.
  (*documentation inherited from `de.croggle.game.achievement.Achievement`*)

6.6.7 `public abstract class PerLevelAchievement extends de.croggle.game.achievement.Achievement`

Description

Achievement for passing certain, specified goals within a level, e.g. placing more than 10 Alligators within one level or 5 eggs hatched within one level.

Constructors

- `public PerLevelAchievement()`

Methods

- `public abstract int requirementsMet()`
  Calculates the index of the stage the achievement has reached, according to the current statistics.
  (*documentation inherited from* `de.croggle.game.achievement.Achievement`)

6.6.8 `public class TimeAchievement extends de.croggle.game.achievement.Achievement`

Description

Achievements which are awarded for reaching certain, specified amounts of time spent playing the game.

Constructors

- `public TimeAchievement()`

Methods

- `public int requirementsMet()`
  Calculates the index of the stage the achievement has reached, according to the current statistics.
  (*documentation inherited from* `de.croggle.game.achievement.Achievement`)

6.7 Package `de.croggle.game.board`

6.7.1 `public class AgedAlligator extends de.croggle.game.board.Alligator`

Description

Aged alligators represent mandatory braces within the Lambda Calculus. They manipulate the order in which the elements are evaluated.

Constructors

- `public AgedAlligator()`
  Creates an aged alligator with no child objects and no parent.
- `public AgedAlligator(Parent parent)`
  Creates an aged alligator with no children and the given parent.

  Parameters

  `parent` the parent this alligator should have

Methods

- `public void accept(BoardObjectVisitor visitor)`
  Accepts a visitor, which is then used for traversing the subtree of the object.

  Parameters

  `visitor` the visitor that tries to access the tree
- `public AgedAlligator copy()`
  Creates and returns a deep copy of the board object.

  Returns

  the deep copy of this object
- `public Parent getParent()`
  Returns the parent object of the internal board object, meaning the parent node in the tree structure.

  Returns

  the parent object

- `public boolean` **`isMovable()`**
  Gets whether the object is movable by the user.

  Returns

  true if the object can be moved, otherwise false

- `public boolean` **`isRemovable()`**
  Gets whether the object is removable by the user.

  Returns

  true if the object can be removed, otherwise false

- `public void` **`setParent(Parent parent)`**
  Sets the given parent as the parent object.

  Parameters

  `parent` the new parent of this object

6.7.2 `public abstract class Alligator extends de.croggle.game.board.Parent implements de.croggle.game.board.InternalBoardObject`

Description

Alligator is the abstract super class of aged and colored alligators. Both have their similar rendering in common (which implies a similar type) and they share the aspect of being parents.
E.g. for the statistics about how many alligators have been transformed, both aged and colored alligators should count and thus need to be assignable to one class of references.

Constructors

- `protected` **`Alligator()`**
  Superconstructor for all alligators.

Methods

- `public abstract void` **`accept(BoardObjectVisitor visitor)`**
  Accepts a visitor which is then used for traversing the subtree of the object.

  Parameters

  `visitor` the visitor that tries to access the tree

- `public abstract Parent` **`getParent`**`()`
  Returns the parent object of this alligator.

  Returns

  the parent alligator

- `public void` **`setParent`**`(Parent parent)`
  Sets the given parent as the parent object.

  Parameters

`parent`	the new parent of this object

6.7.3 `public class AlligatorOverflowException extends java.lang.Exception`

Description

The exception is thrown whenever a lambda term contains more than 300 InternalBoar-dObjects. This happens while adding alligators or eggs to the working set, e.g. in the sandbox mode.

Constructors

- `public` **`AlligatorOverflowException`**`()`
  Creates a new instance of the exception with the default constructor.

- `public` **`AlligatorOverflowException`**`(String message)`
  Creates a new instance of the exception with the given error message.

  Parameters

`message`	a message describing the cause of the exception that occured

6.7.4 `public class Board extends de.croggle.game.board.Parent implements de.croggle.game.board.BoardObject`

Description

Root object of every alligator term. This class acts as the root of the tree structure used for modelling the lambda terms.

Constructors

- `public Board()`
  Creates a new board with no children.

Methods

- `public void accept(BoardObjectVisitor visitor)`
  Accepts a visitor which is then used for traversing the object's subtree.

  Parameters

  `visitor` the visitor that tries to access the tree

- `public Board copy()`
  Creates and returns a deep copy of the board object.

  Returns

  the deep copy

6.7.5 `public interface interface BoardObject`

Description

An interface for any object which resides on the board.

Methods

- `public void accept(BoardObjectVisitor visitor)`
  Accepts a visitor which is then used for traversing the object's subtree.

  Parameters

  `visitor` the visitor that tries to access the tree

- `public BoardObject copy()`
  Creates and returns a deep copy of the board object.

  Returns

  the deep copy

6.7.6 `public class ColoredAlligator extends de.croggle.game.board.Alligator implements de.croggle.game.board.ColoredBoardObject`

Description

Colored alligators represent lambda abstractions in the Lambda Calculus. The color of the alligator is the variable which is bound by the abstraction.

Constructors

- `public ColoredAlligator(Color c)`
  Creates a new ColoredAlligator with the specified color. The color hereby serves as the name of variables bound by this abstraction in the Lambda Calculus. The ColoredAlligator is created as a recolorable board object by this constructor.

  Parameters

`c`	the color this alligator has

- `public ColoredAlligator(Color c, boolean recolorable)`
  Creates a new ColoredAlligator with the specified color and the permission value if the object is recolorable or not. The color hereby serves as the name of variables bound by this abstraction in the Lambda Calculus.

  Parameters

`c`	the color this alligator has
`recolorable`	whether the ColoredAlligator is recolorable (true) or not (false)

Methods

- `public void accept(BoardObjectVisitor visitor)`
  Accepts a visitor which is then used for traversing the subtree of the object.

  Parameters

`visitor`	the visitor that tries to access the tree

- `public ColoredAlligator copy()`
  Creates and returns a deep copy of the board object.

  Returns

  the deep copy

- `public Color getColor()`
  Returns the color that represents the name of a variable on the board.

Returns

the current color of the colored alligator

- `public Parent` **`getParent`**`()`
  Returns the parent object of this alligator.

  Returns

  the parent alligator

- `public boolean` **`isMovable`**`()`
  Returns whether the object can be moved by the user.

  Returns

  true if the object can be moved, otherwise false

- `public boolean` **`isRecolorable`**`()`
  Returns whether the color of the egg can be changed by the user.

  Returns

  true if the object can be recolored, otherwise false

- `public boolean` **`isRemovable`**`()`
  Returns whether the user can remove this alligator from the board.

  Returns

  true if the object can be removed, otherwise false

- `public void` **`setColor`**`(Color c)`
  Sets the color of the alligator. In placement mode: Set color only if it is marked as recolorable.

  Parameters

  `c` the new color for the colored alligator

- `public void` **`setParent`**`(Parent parent)`
  Sets the given parent as the parent object.

  Parameters

  `parent` the new parent of this object

6.7.7 `public interface interface ColoredBoardObject`

Description

Interface of all colorable and recolorable board objects. In other words, this interface adds naming functionality to syntax tree elements.

Methods

- `public Color getColor()`
  Returns the color that represents a variable's name on the board.

  Returns

  the current color of the board object

- `public boolean isRecolorable()`
  Returns whether the color of the board object can be changed by the user.

  Returns

  true if the object can be recolored, otherwise false

- `public void setColor(Color c)`
  Sets the color of the board object. In placement mode: Set only if it is marked as recolorable.

  Parameters

  `c` the new color for the board object

6.7.8 `public class Egg implements de.croggle.game.board.InternalBoardObject, de.croggle.game.board.ColoredBoardObject`

Description

An egg represents a variable within the Lambda Calculus. If the guarding alligator eats something, the eaten thing will hatch from the egg.

Constructors

- `public Egg(Color c)`
  Creates a new egg with the specified color. The color hereby serves as the name of the variable this egg represents in the Lambda Calculus. The egg is created as a recolorable board object by this constructor.

  Parameters

  `c` the color this egg has.

- `public Egg(Color c, boolean recolorable)`
  Creates a new egg with the specified color and the permission value if the object is recolorable or not. The color hereby serves as the name of the variable this egg represents in the Lambda Calculus.

  Parameters

`c`	the color this egg has.
`recolorable`	whether the egg is recolorable (true) or not (false)

Methods

- `public void accept(BoardObjectVisitor visitor)`
  Accepts a visitor, which is then used for traversing the subtree of the object.

  Parameters

`visitor`	the visitor that tries to access the tree

- `public Egg copy()`
  Creates and returns a deep copy of the board object.

  Returns

  the deep copy

- `public Color getColor()`
  Returns the color that represents a variable's name on the board.

  Returns

  the current color of the egg

- `public Parent getParent()`
  Returns the parent object of this egg.

  Returns

  the parent alligator

- `public boolean isMovable()`
  Returns whether the egg can be moved by the user.

  Returns

  true if the egg can be moved, otherwise false

- `public boolean isRecolorable()`
  Returns whether the color of the egg can be changed by the user.

  Returns

  true if the object can be recolored, otherwise false

- `public boolean` **`isRemovable`**`()`
  Returns whether the egg can be removed by the user.

  Returns

  true if the object can be removed, otherwise false

- `public void` **`setColor`**`(Color c)`
  Sets the color of the egg. In placement mode: Set only if it is marked as recolorable.

  Parameters

`c`	the new color for the egg

- `public void` **`setParent`**`(Parent parent)`
  Sets the given parent as the parent object.

  Parameters

`parent`	the new parent of this object

6.7.9 `public class IllegalBoardException extends java.lang.Exception`

Description

The exception is thrown when the alligator constellation does not represent a correct lambda term after the simulation has been started.

Constructors

- `public` **`IllegalBoardException`**`()`
  Creates a new instance of the exception with the default constructor.

- `public` **`IllegalBoardException`**`(String message)`
  Creates a new instance of the exception with the given error message.

  Parameters

`message`	a message describing the cause of the exception that occured

6.7.10 `public interface interface InternalBoardObject extends de.croggle.game.board.BoardObject`

Description

Special type of BoardObject whose specific attribute is that its not the uppermost BoardObject which means it must have a parent.

Methods

- `public Parent` **`getParent`**`()`
  Returns the parent object of the internal board object, meaning the parent node in the tree structure.

  Returns

  the parent object

- `public boolean` **`isMovable`**`()`
  Returns whether the object can be moved by the user.

  Returns

  true if the object can be moved, otherwise false

- `public boolean` **`isRemovable`**`()`
  Returns whether the object can be removed by the user.

  Returns

  true if the object can be removed, otherwise false

- `public void` **`setParent`**`(Parent parent)`
  Sets the given parent as the parent object.

  Parameters

`parent`	the new parent of this object

6.7.11 `public abstract class Parent`

Description

Parent is an abstract class to model the functions board objects - which can be parents of families - must have.

Constructors

- `protected` **`Parent()`**
  Superconstructor of all parents. Creates a parent with no children.

Methods

- `public void` **`addChild`**`(InternalBoardObject child)`
  Adds a child to the family of the parent.

  Parameters

  `child` the child which should be added to the family of the parent

- `public Iterator<InternalBoardObject>` **`getIterator`**`()`
  Returns an iterator for the children list.

  Returns

  the iterator

- `public InternalBoardObject` **`getNextChild`**`(InternalBoardObject child)`
  Returns the child next to (to the right of) the one given as a parameter.

  Parameters

  `child` the child whose successor should be returned

  Returns

  the child which is the next one in the list after the given child

- `public boolean` **`isLastChild`**`(InternalBoardObject child)`
  Checks whether the given InternalBoardObject is the last/rightmost child in the list of the parent.

  Parameters

  `child` the child which should be checked on

  Returns

  returns true if the given child is the last child in the list, false otherwise

- `public boolean` **`removeChild`**`(InternalBoardObject child)`
  Removes a child from the family of the parent.

  Parameters

`child`	the child which should be removed from the family of the parent

  Returns

  whether the removal was successful

- `public boolean` **`replaceChildWith`**`(InternalBoardObject child, InternalBoardObject replaceChild)`
  Replaces a child object with another one. If the given child is not found, nothing is replaced and false is returned.

  Parameters

`child`	the child to replace
`replaceChild`	the child that replaces the current child

  Returns

  true on success, false otherwise

6.8 Package `de.croggle.game.event`

6.8.1 `public interface interface AgedAlligatorVanishesListener`

Description

Interface for listeners specifically listening to the onAgedAlligatorVanishes event. This event is produced when a simulator removes any instance of an aged alligator from its associated board. The class is kept general for both the rendered animation and the vanished alligator statistics.

Methods

- `public void` **`onAgedAlligatorVanishes`**`(AgedAlligator alligator)`
  Receive an alligator vanishes event for further processing. E.g. the statistics manager can count how many alligators have vanished/been transformed on the board in a game.

  Parameters

`alligator`	the vanishing alligator

6.8.2 `public interface interface BoardEventListener extends de.croggle.game.event.ReplaceEventListener, de.croggle.game.event.ObjectRecoloredListener, de.croggle.game.event.EatEventListener, de.croggle.game.event.BoardRebuiltEventListener, de.croggle.game.event.AgedAlligatorVanishesListener`

Description

Interface for aggregating all types of listeners for events concerning board objects, so that implementing classes must only implement this interface.

6.8.3 `public class BoardEventMessenger`

Description

The location in which listeners are able to register and unregister themselves so they would recieve further notifications, e.g. when an object has been recolored. Objects of this class can easily be passed to methods, so that these can trigger events.

Constructors

- `public` **`BoardEventMessenger`**`()`

Methods

- `public void` **`notifyAgedAlligatorVanishes`**`(AgedAlligator alligator)`
  Sends an `onAgedAlligatorVanishes` event to all registered listeners.

  Parameters

`alligator`	the alligator that has vanished

- `public void` **`notifyBoardRebuilt`**`(Board board)`
  Sends an `onBoardRebuilt` event to all registered listeners.

  Parameters

`board`	the board which was rebuilt

- `public void` **`notifyEat`**`(ColoredAlligator eater, InternalBoardObject eatenFamily)`
  Sends an `onEat` event to all registered listeners.

  Parameters

`eater`	the colored alligator that has eaten
`eatenFamily`	the family which was eaten

- `public void` **`notifyObjectRecolored`**`(InternalBoardObject recoloredObject)`
  Sends an `onObjectRecolored` event to all registered listeners.

  Parameters

`recoloredObject`	the object which was recolored

- `public void` **`notifyReplace`**`(Egg replacedEgg, InternalBoardObject bornFamily)`
  Sends an `onReplace` event to all registered listeners.

  Parameters

`replacedEgg`	the egg that was replaced
`bornFamily`	the family that hatched out of the egg

- `public void` **`register`**`(BoardEventListener listener)`
  Registers a new listener to listen for board events sent via this messenger. The listener will receive all future events, until it is unregistered.

  Parameters

`listener`	the listener to register

- `public void` **`unregister`**`(BoardEventListener listener)`
  Unregisters a listener so that it won't receive any events sent via this messenger. If the listener isn't registered, this method has no effect.

  Parameters

`listener`	the listener to unregister

6.8.4 `public interface interface BoardRebuiltEventListener`

Description

Interface for listeners specifically listening to the onBoardRebuilt event. This event is produced when a simulator does something that requires the complete renewal of the elements shown on the board, e.g. undoing the last step or reseting to the initial board upon entering the simulation mode.

Methods

- `public void` **`onBoardRebuilt`**`(Board board)`
  Receive a board rebuilt event for further processing. E.g. the renderer can determine by accepting a board rebuilt event that the elements shown on the board have to be renewed.

  Parameters

`board`	the board that has to be rebuild

6.8.5 `public interface interface EatEventListener`

Description

Interface for listeners specifically listening to the onEat event. This event is produced when a simulator applies the begin of the eating rule. That is, when a subtree (an alligator with its family or alternatively just an egg) is "eaten" by another alligator.

Methods

- `public void` **`onEat`**`(ColoredAlligator eater, InternalBoardObject eatenFamily)`
  Receive an eat event for further processing. E.g. the renderer can determine by accepting an eat event where an eat animation has to be played.

  Parameters

`eater`	the alligator "eating" the EatenFamily
`eatenFamily`	the parent of all eaten objects, which is being eaten himself

6.8.6 `public interface interface ObjectRecoloredListener`

Description

Interface for listeners specifically listening to the onObjectRecolored event. This event is produced when a simulator performs a recoloring on an internal board object on the board. E.g. this can be caused by the player or alternatively when an alpha conversion occurs.

Methods

- `public void` **`onObjectRecolored`**`(InternalBoardObject recoloredObject)`
  Receive an object recolored event for further processing. E.g. the renderer can determine by accepting an eat event where a board object has to be recolored.

  Parameters

`recoloredObject`	the board object whose color changed

6.8.7 `public interface interface ReplaceEventListener`

Description

Interface for listeners specifically listening to the onReplace event. This event is produced after a simulator has realized the end of the eating rule. That is, when a copy of a subtree (an alligator with its family or alternatively just an egg) “hatched out“ of an egg. Event listeners can assume that the replacement has already completely taken place. That means, that bornFamily has its new parent set and the replacedEgg is not in the list of childs of its parent any more.

Methods

- `public void` **`onReplace`**`(Egg replacedEgg, InternalBoardObject bornFamily)`
  Receive an object replaced event for further processing. E.g. the renderer can determine by accepting a replaced event where an egg’s rendering needs to be replaced with a new family.

  Parameters

`replacedEgg`	the egg which hatches out to become the born family
`bornFamily`	the family which will emerge from an egg

6.9 Package `de.croggle.game.level`

6.9.1 `public class ColorEditLevel extends de.croggle.game.level.Level`

Description

A special type of level in which the player has to change the color of the given elements in order for the simulation to reach a certain outcome.

Attributes

- `ColorController` **`colorController`**

Constructors

- `public` **`ColorEditLevel()`**

6.9.2 `public abstract class Level`

Description

This class represents the concept of a level within the game.

Constructors

- `public` **`Level()`**
  Creates an empty level with the default values.

Methods

- `public int` **`getAbortSimulationAfter()`**
  Gets number of steps the simulation runs before it is aborted.

  Returns

  the number of steps the simulation runs

- `public Animation` **`getAnimation()`**
  Gets the path to the animation of the level if it has one.

  Returns

  the path to the animation of the level

- `public String` **`getDescription`**`()`
  Gets the description of the level.

  Returns

  the description of the level

- `public Board` **`getGoalBoard`**`()`
  Gets the board, which has to be reached to win the level.

  Returns

  the board which is the goal of the level

- `public String` **`gethint`**`()`
  Gets the path to the hint of the level.

  Returns

  the path to the hint of the level

- `public Board` **`getInitialBoard`**`()`
  Gets the board the level starts with.

  Returns

  the initial board

- `public int` **`getLevelIndex`**`()`
  Gets the index of the level in the level package.

  Returns

  the index of the level

- `public int` **`getPackageIndex`**`()`
  Gets the index of the level package this level belongs to.

  Returns

  the index of the level package

- `public Color[]` **`getUserColors`**`()`
  Gets the colors usable by the user.

  Returns

  an array of colors

- `public boolean` **`hasAnimation`**`()`
  Checks whether this level has a simulation or not.

Returns

true if the level has a simulation, otherwise false

6.9.3 `public class LevelController`

Description

Controls the content of a level package.

Constructors

- `public LevelController()`
  Creates the controller with an empty list of levels.
- `public LevelController(List<Level> levels)`
  Creates the controller with a given list of levels to manage.

  Parameters

`levels`	the list of levels the controller should hold

- `public LevelController(int packageIndex)`
  Creates the controller with the given package index. It will manage the levels from the level package defined by `packageIndex`.

  Parameters

`packageIndex`	the index of the package whose levels should be controlled

Methods

- `public Level getLevel(int levelIndex)`
  Returns the level specified by the given index. The index must be between 0 and `getPackageSize() - 1`.

  Parameters

`levelIndex`	the index of the level that should be returned

  Returns

  the desired level

- `public int getPackageIndex()`
  Returns the package index of the package of the level the controller currently holds.

Returns

the package index

- `public int` **`getPackageSize()`**
  Returns the size of the package, i.e. how many levels the controller holds.

  Returns

  the package size

6.9.4 `public class LevelPackage`

Description

Compilation of several, thematically linked levels.

Constructors

- `public` **`LevelPackage()`**
  Creates a level package with default values.
- `public` **`LevelPackage(int levelPackageId)`**
  Creates a level package with the given id. All other fields have the default values.

  Parameters

`levelPackageId`	the folder id of the levelPackage

Methods

- `public String` **`getDescription()`**
  Returns a textual description of the package.

  Returns

  the package description
- `public String` **`getEmblemPath()`**
  Returns the file path to the graphical representation of the package.

  Returns

  the emblem path
- `public LevelController` **`getLevelController()`**
  Gets the level controller which is responsible for handling the levels within the level package.

Returns

the level controller one must use to handle the levels within the level package

- `public int` **`getLevelPackageId`**`()`
  Returns the unique identifier of the package, which is defined as the name of the folder the package represents.

  Returns

  the package id

- `public String` **`getName`**`()`
  Returns the packages name.

  Returns

  the package name

- `public int` **`getSize`**`()`
  Returns the amount of levels in this package.

  Returns

  the size of the package

- `public boolean` **`hasAnimation`**`()`
  Indicates whether the package has an animation, which is shown when it is started for the first time (like a little story that is told).

  Returns

  true, if the package has an animation defined, false otherwise

6.9.5 `public class LevelPackagesController`

Description

Controls the overview over the different level packages.

Constructors

- `public` **`LevelPackagesController`**`(AlligatorApp game)`
  Creates a new controller with no packages attached.

  Parameters

  `game` the backreference to the central game object

6.9.6 `public class LoadLevelHelper`

Description

Encapsulates the functionality needed for instantiating a level/game from the respective JSON file. Therefore it removes a larger portion of program logic from the `LevelController`, which in turn delegates requests for level instantiation to this class' instantiate method.

Methods

- `static Level` **`instantiate`**`(int packageIndex, int levelIndex)`
  Called to load a new level. With both the package index and the level index it is possible to distinctively indentify the required level.

  Parameters

`packageIndex`	specifies the level package from which the level is supposed to be loaded
`levelIndex`	the id of the level within the package

  Returns

  the level denoted by the given indices/identifiers

6.9.7 `public class MultipleChoiceLevel extends de.croggle.game.level.Level`

Description

A special type of level in which the player has to choose from several options, one of which is the correct one.

Constructors

- `public` **`MultipleChoiceLevel`**`()`

Methods

- `public boolean` **`hasAnimation`**`()`
  Checks whether this level has a simulation or not.
  (*documentation inherited from* `de.croggle.game.level.Level`)
- `public boolean` **`validateAnswer`**`(int selection)`
  Method to check whether the given answer was the correct one.

Parameters

`selection`	the index of the selected answer

Returns

true if the answer was right, false otherwise

6.9.8 `public class TermEditLevel extends de.croggle.game.level.Level`

Description

A special type of level in which the player has to position and color alligators and eggs into a constellation that evaluates into the given goal term.

Attributes

- `ColorController` **`colorController`**

Constructors

- `public` **`TermEditLevel()`**

Methods

- `public boolean` **`hasAnimation()`**
  Checks whether this level has a simulation or not.
  (*documentation inherited from* `de.croggle.game.level.Level`)

6.10 Package `de.croggle.game.profile`

6.10.1 `public class Profile`

Description

Represents a profile with its settings and statistics.

Constructors

- `public` **`Profile`**`(String name, String picturePath)`
  Creates a new profile with default settings and statistics.

  Parameters

`name`	the name of the profile
`picturePath`	the path to the profiles' picture

Methods

- `public String` **`getName`**`()`
  Get the profile's name.

  Returns

  the profile's name

- `public String` **`getPicturePath`**`()`
  Get the path to where the profile's picture is stored.

  Parameters

`name`	the path to the location where the profile's picture is stored

- `public Setting` **`getSetting`**`()`
  Get the profile's setting.

  Parameters

`name`	the profile's setting

- `public Statistic` **`getStatistic`**`()`
  Get the profile's statistic.

  Parameters

`name`	the profile's statistic

- `public void` **`setName`**`(String name)`
  Set the profile's name.

  Parameters

`name`	the new name of the profile

- `public void` **`setPicturePath`**`(String picturePath)`
  Set the profile's picture path.

  Parameters

`name`	The new path to the profile's picture

- `public void` **`setSetting`**`(Setting setting)`
  Set the profile's setting.

  Parameters

`name`	the new setting of the profile

- `public void` **`setStatistic`**`(Statistic statistic)`
  Set the profile's statistic.

  Parameters

`name`	the new setting of the profile

6.10.2 `public class ProfileController`

Description

A controller made to encapsulate the management of profiles. There is always one of six possible profiles active.

Constructors

- `public` **`ProfileController`**`(AlligatorApp game)`
  Creates a new profile controller. On initialization the active profile is set to null.

  Parameters

`game`	the backreference to the central game object

Methods

- `public void` **`changeCurrentProfile`**`(String newProfileName)`
  Sets the profile identified by the given profile name as the active profile. The profile that was active before needs to be entirely saved before calling this method.

Parameters

`newProfileName`	the string identifying the new profile

Throws

`IllegalArgumentException`	when there is no saved profile identified by the given profile name

- `public void` **`createNewProfile`**`(String name, String picturePath)`
  Creates a new profile with the given attributes and sets it as the active profile. Also writes it to the database.

  Parameters

`name`	the unique name of the owner of the new profile
`picturePath`	the picture path to the picture associated with the new profile

  Throws

`IllegalArgumentException`	if there already is a profile identified by the given name
`ProfileOverflowException`	if there already are six profiles registered

- `public void` **`deleteCurrentProfile`**`()`
  Completely removes the currently active profile (also from the database). After deletion, there is no active profile.

- `public void` **`editCurrentProfile`**`(Profile profile)`
  Replaces the active profile entirely by the given new one. There must be an active profile set (not null).

  Parameters

`profile`	the profile which should replace the active profile

  Throws

`IllegalArgumentException`	when the given profile is null or its name already identifies another profile

- `public List<Profile>` **`getAllProfiles`**`()`
  Gets a list of all profiles.

  Returns

  the list of all saved profiles

- `public boolean` **`isValidUserName`**`(String newUserName)`
  Tests and returns if a string supposed to be a new profile's identifier is valid,

meaning if it contains at least one character and is not already identifier of another profile.

Parameters

`newUserName`	The string to be tested

Returns

true, if new username is a valid profile name, false otherwise

6.10.3 `public class ProfileOverflowException extends java.lang.Exception`

Description

The exception is thrown whenever there are more than the 6 possible profiles.

Constructors

- `public ProfileOverflowException()`
  Creates a new instance of the exception with the default constructor.
- `public ProfileOverflowException(String message)`
  Creates a new instance of the exception with the given error message.

  Parameters

`message`	a message describing the cause of the exception that occured

6.11 Package `de.croggle.game.visitor`

6.11.1 `public interface interface BoardObjectVisitor`

Description

A visitor for traversing trees of BoardObjects. It visits a node at first and then each of it's children from left to right.

Methods

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.

  Parameters

  `alligator` the aged alligator which is visited

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.

  Parameters

  `board` the board which is visited

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.

  Parameters

  `alligator` the colored alligator which is visited

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.

  Parameters

  `egg` the egg which is visited

6.11.2 `public class CollectBoundColorsVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor for collecting all the colors of alligators in a family. This is equivalent to the set of variables which are bound in a given subterm.

Methods

- `public static Color[]` **`collect`**`(BoardObject family)`
  Returns the set of colors of alligators in the given family.

  Parameters

  `family` the family to examine

  Returns

  the set of bound colors

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

6.11.3 `public class CollectFreeColorsVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor for collecting all the colors of eggs with no matching alligator above them. This is equivalent to the set of variables which occur free in a given subterm.

Methods

- `public static Color[]` **`collect`**`(BoardObject family)`
  Returns the set of colors of eggs with no matching alligator above them in the given family.

  Parameters

`family`	the family to examine

  Returns

  the set of free colors

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

6.11.4 `public class CountBoardObjectsVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor for counting the number of objects in a family.

Methods

- `public static int` **`count`**`(BoardObject family)`
  Count the number of objects in a family.

  Parameters

  `family` the family whose members should be counted

  Returns

  the number of family members

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from `de.croggle.game.visitor.BoardObjectVisitor`*)

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.
  (*documentation inherited from `de.croggle.game.visitor.BoardObjectVisitor`*)

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from `de.croggle.game.visitor.BoardObjectVisitor`*)

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from `de.croggle.game.visitor.BoardObjectVisitor`*)

6.11.5 `public class FindEatingVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor for finding a colored alligator which can eat a family next to it.

Methods

- `public static ColoredAlligator` **`findEater`**`(Board board)`
  Search a colored alligator which can eat a family next to it.

  Parameters

  `board` the board in which colored alligators should be searched

  Returns

  the eating alligator if one was found, otherwise null

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

6.11.6 `public class FlattenVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor to flatten the syntax tree. Use the method “flatten“ to receive a “flat“ array of BoardObjects. Useful if sequentially iterating through all objects in the tree is needed to be achieved.

Methods

- `public BoardObject[] flatten(BoardObject tree)`
  Systematically travels the tree and adds each element part of it **one single time** to the array that is returned. Useful if sequentially iterating through all objects in the tree is needed to be achieved.

  Parameters

  `tree` The alligator tree to be flattened.

  Returns

  A list with all elements in the given tree.

- `public void visitAgedAlligator(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void visitBoard(Board board)`
  Called when the board is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void visitColoredAlligator(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void visitEgg(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

6.11.7 `public class RecolorVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor for replacing occurences of one color in a family with another color.

Methods

- `public static void` **`recolor`**`(BoardObject family, Color oldColor, Color newColor, BoardEventMessenger boardMessenger)`
  Recolor all alligators and eggs in `family` which have the color `oldColor` with `newColor`.

  Parameters

`family`	the family to recolor
`oldColor`	the color to replaced
`newColor`	the color used for replacing the old color
`boardMessenger`	the messenger used for notifying listeners about the recoloring

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

6.11.8 `public class RemoveAgedAlligatorsVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor looking for aged alligators, which are not nessesary because they have only one children.

Methods

- `public static void` **`remove`**`(BoardObject family, BoardEventMessenger boardMessenger)`
  Removes all old alligators which are not necessary.

  Parameters

`family`	the family in which old alligators should be removed
`boardMessenger`	the messenger used for notifying listeners about removed alligators

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

6.11.9 `public class ReplaceEggsVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor replacing eggs of a certain color with copies of a given family (subtree).

Methods

- `public static void` **`replace`**`(ColoredAlligator parent, InternalBoardObject bornFamily, BoardEventMessenger boardMessenger, ColorController colorController)`
  Replaces all eggs below `parent`, which share it's color, with a copy of `bornFamily`.
  Correct recoloring of the newly inserted families is also performed, if necessary.
  When an egg is replaced, an onEat event is sent through `boardMessenger`.

  Parameters

`parent`	the colored alligator whose child eggs should be replaced
`bornFamily`	the family with which eggs are replaced
`boardMessenger`	the messenger used for sending events when eggs are replaced
`colorController`	the color controller used for recoloring

  Throws

`ColorOverflowException`	if recoloring occurs and there is no color available

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

6.11.10 `public class ValidationVisitor implements de.croggle.game.visitor.BoardObjectVisitor`

Description

A visitor for checking whether the given Board represents a valid term within the lambda calculus (whether the evaluation is possible or not).

Methods

- `public static boolean` **`isValid`**`(BoardObject family)`
  Checks whether the given family represents a valid term within the lambda calculus.

  Parameters

`family`	the family to check for validity

  Returns

  true if the family is valid, false otherwise

- `public void` **`visitAgedAlligator`**`(AgedAlligator alligator)`
  Called when an aged alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitBoard`**`(Board board)`
  Called when the board is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitColoredAlligator`**`(ColoredAlligator alligator)`
  Called when a colored alligator is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

- `public void` **`visitEgg`**`(Egg egg)`
  Called when an egg is visited.
  (*documentation inherited from* `de.croggle.game.visitor.BoardObjectVisitor`)

6.12 Package `de.croggle.ui`

6.12.1 `public class StyleHelper`

Description

Singleton to manage Styles.

Methods

- `ImageButton.ImageButtonStyle` **`getImageButtonStyle()`**
  Gets the style of the image button.

  Returns

  the image button's style

- `ImageTextButton.ImageTextButtonStyle` **`getImageTextButtonStyle()`**
  Gets the style of the image text button.

  Returns

  the image text button's style

- `public StyleHelper` **`getInstance()`**
  Gets the current instance.

  Returns

  the current instance

- `Label.LabelStyle` **`getLabelStyle()`**
  Gets the style of the label.

  Returns

  the label's style

- `public Skin` **`getSkin()`**
  Gets the used skin.

  Returns

  the skin

- `TextButton.TextButtonStyle` **`getTextButtonStyle()`**
  Gets the style of the text button.

Returns

the text button's style

6.13 Package `de.croggle.ui.actors`

6.13.1 `public class ObjectBar extends com.badlogic.gdx.scenes.scene2d.Actor`

Description

The bar to drag alligators and eggs from onto the screen.

Constructors

- `public ObjectBar()`
  Creates an object bar with the ui elements the user can drag to the screen per default.

6.14 Package `de.croggle.ui.renderer`

6.14.1 `public class AgedAlligatorActor extends de.croggle.ui.renderer.ParentActor`

Description

An actor used for representing an aged alligator.

Constructors

- `public AgedAlligatorActor()`
  Creates a new actor.

Methods

- `public void act(float delta)`
  Updates the actor based on time.
  (*documentation inherited from `de.croggle.ui.renderer.ParentActor`*)

- `public void draw(SpriteBatch batch, float parentAlpha)`
  Draws the actor. The sprite batch is configured to draw in he parent's coordinate system.
  (*documentation inherited from `de.croggle.ui.renderer.ParentActor`*)

6.14.2 `public class BoardActor extends de.croggle.ui.renderer.ParentActor implements de.croggle.game.event.BoardEventListener`

Description

An actor used for representing an alligator constellation.

Constructors

- `public BoardActor()`
  Creates a new actor.

Methods

- `public void act(float delta)`
  Updates the actor based on time.
  (*documentation inherited from `de.croggle.ui.renderer.ParentActor`*)

- `public void` **`draw`**`(SpriteBatch batch, float parentAlpha)`
  Draws the actor. The sprite batch is configured to draw in he parent's coordinate system.
  (*documentation inherited from* `de.croggle.ui.renderer.ParentActor`)

- `public void` **`onAgedAlligatorVanishes`**`(AgedAlligator alligator)`
  Visualizes the disappearance of an aged alligator on the board.

  Parameters

`alligator`	the alligator which disappeared

- `public void` **`onBoardRebuilt`**`(Board board)`
  Completely rebuilds the board as it is seen on the screen.

  Parameters

`board`	the board that is going to replace the board that was seen previously

- `public void` **`onEat`**`(ColoredAlligator eater, InternalBoardObject eatenFamily)`
  Visualizes the process of one alligator eating another and its children, or just an egg, on the board.

  Parameters

`eater`	the alligator which eats the other alligator
`eatenFamily`	the family which is eaten by the other alligator

- `public void` **`onObjectRecolored`**`(InternalBoardObject recoloredObject)`
  Visualizes the recoloring of an object on the board.

  Parameters

`recoloredObject`	the object that has been recolored

- `public void` **`onReplace`**`(Egg replacedEgg, InternalBoardObject bornFamily)`
  Visualizes the process of replacing an egg within a family with the family the protecting alligator has eaten.

  Parameters

`replacedEgg`	the hatching egg
`bornFamily`	the family that hatches from that egg

6.14.3 `public abstract class BoardObjectActor extends com.badlogic.gdx.scenes.scene2d.Actor`

Description

An actor used for representing a board object.

Constructors

- `public BoardObjectActor()`
  The superconstructor for all board object actors.

6.14.4 `public class ColoredAlligatorActor extends de.croggle.ui.renderer.ParentActor`

Description

An actor used for representing a colored alligator.

Constructors

- `public ColoredAlligatorActor()`
  Creates a new actor.

Methods

- `public void act(float delta)`
  Updates the actor based on time.
  (*documentation inherited from* `de.croggle.ui.renderer.ParentActor`)

- `public void draw(SpriteBatch batch, float parentAlpha)`
  Draws the actor. The sprite batch is configured to draw in he parent's coordinate system.
  (*documentation inherited from* `de.croggle.ui.renderer.ParentActor`)

- `public void enterDyingState()`
  Signals the actor to enter the dying rendering state. That is, an alligator with a specific color, mouth closed and turned on its back. Will initiate a transition animation from mouth open to closed if it was open previously. Also turns the alligator around 180 degree, if it wasn't in this state before.

- `public void enterEatingState()`
  Signals the actor to enter the eating rendering state. That is, an alligator with a specific color, mouth opened. Will initiate a transition animation from mouth closed to open if it was closed previously.

- `public void` **`enterNormalState`**`()`
  Signals the actor to (re-)enter the normal rendering state. That is, an alligator with a specific color, mouth closed. Will initiate a transition animation from mouth open to closed if it was open previously.

6.14.5 `public class EggActor extends de.croggle.ui.renderer.BoardObjectActor`

Description

An actor used for representing an egg.

Constructors

- `public` **`EggActor`**`()`
  Creates a new actor.

Methods

- `public void` **`act`**`(float delta)`
  Updates the actor based on time.

  Parameters

`delta`	Time in seconds since the last update.

- `public void` **`draw`**`(SpriteBatch batch, float parentAlpha)`
  Draws the actor. The sprite batch is configured to draw in he parent's coordinate system.

  Parameters

`batch`	The sprite batch specifies where to draw into.
`parentAlpha`	the parent's alpha value

- `public void` **`enterHatchingState`**`()`
  Signals the actor to enter the hatching rendering state. That is, scattered eggshell with the specific color. Will initiate a transition animation from a normal egg to the broken eggshell.

- `public void` **`enterNormalState`**`()`
  Signals the actor to (re-)enter the normal rendering state. That is, an egg with a specific color.

6.14.6 `public abstract class ParentActor extends de.croggle.ui.renderer.BoardObjectActor`

Description

An actor used for representing a parent object.

Constructors

- `public ParentActor()`
  Superconstructor for all render actors.

Methods

- `public void act(float delta)`
  Updates the actor based on time.

  Parameters

`delta`	time in seconds since the last update

- `public void draw(SpriteBatch batch, float parentAlpha)`
  Draws the actor. The sprite batch is configured to draw in he parent's coordinate system.

  Parameters

`batch`	the sprite batch specifies where to draw into
`parentAlpha`	the parent's alpha value

6.15 Package `de.croggle.ui.screens`

6.15.1 `public abstract class AbstractScreen implements com.badlogic.gdx.Screen`

Description

Abstract screen, with all the basic things a screen needs.

Constructors

- `public` **`AbstractScreen`**`(AlligatorApp game)`
  Superconstructor for all screens. Initializes everything they share, e.g. their stage.

  Parameters

`game`	the backreference to the central game

Methods

- `public void` **`dispose`**`()`
  Called in order to cause the screen to release all resources held.

- `public void` **`hide`**`()`
  Called when this screen should no longer be the game's current screen.

- `public void` **`pause`**`()`
  Called when this screen is paused. A screen is paused before it is destroyed, when the user pressed the Home button or e.g. an incoming call happens.

- `public void` **`render`**`(float delta)`
  Called when the screen should render itself.

- `public void` **`resize`**`(int width, int height)`
  Called when the application is resized. This can happen at any point during a non-paused state but will never happen before a call to create().

  Parameters

`width`	the width, which the newly resized screen will have.
`height`	the height, which the newly resized screen will have.

- `public void` **`resume`**`()`
  Called in order to move the screen back from its paused state.

- `public void` **`show`**`()`
  Called when this screen should be the game's current screen.

6.15.2 `public class AchievementScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen listing the achievements, both achieved and unachieved, in a sorted way. For reference see “Pflichtenheft 10.5.8 / Abbildung 17”.

Constructors

- `public` **`AchievementScreen`**`(AchievementController achievement)`
  Creates the achievement overview screen that uses the given achievement controller to display the current achievement progress.

  Parameters

`achievment`	the achievement controller

6.15.3 `public class LevelPackagesScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen, in which one can select the levelpackage. For reference see “Pflichtenheft 10.5.2 / Abbildung 10”.

Constructors

- `public` **`LevelPackagesScreen`**`(LevelPackagesController controller)`
  Creates the level package overview screen that uses the level package controller to display the different level packages.

  Parameters

`controller`	the level package controller

6.15.4 `public class LevelsOverviewScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen, in which one can choose to play different levels within the selected levelpackage. For reference see "Pflichtenheft 10.5.3 / Abbildung 11".

Constructors

- `public` **`LevelsOverviewScreen`**`(LevelController controller)`
  Creates the level overview screen that uses the given level controller to display the levels within the selected level package.

  Parameters

`controller`	the level controller

6.15.5 `public class LevelTerminatedScreen extends de.croggle.ui.screens.AbstractScreen`

Description

First screen seen after completing a level. For reference see "Pflichtenheft 10.5.6 / Abbildung 15".

Constructors

- `public` **`LevelTerminatedScreen`**`(GameController controller)`
  Creates the level terminated screen that is shown to the player after the completion of a level.

  Parameters

`controller`	the game controller, who is responsible for the completed level

6.15.6 `public class MainMenuScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen, which shows the central menu one uses to navigate into every other point of the application. For reference see "Pflichtenheft 10.5.1 / Abbildung 9".

Constructors

- `public MainMenuScreen(AlligatorApp app)`
  Creates the main menu screen from whom the player can navigate into the different parts of the app.

  Parameters

  `app` the instance of alligator app, from which everything is connected

6.15.7 `public class MultipleChoiceScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen which is shown while playing a multiple choice level.

Constructors

- `public MultipleChoiceScreen(GameController controller)`
  Creates the base screen of a multiple choice level, which is shown to the player upon entering a multiple choice level.

  Parameters

  `controller` the game controller responsible for the multiple choice level

6.15.8 `public class PlacementModeScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen within which the player can manipulate the board by moving alligators and eggs. For reference see "Pflichtenheft 10.5.4 / Abbildungen 12 und 1".

Constructors

- `public PlacementModeScreen(GameController controller)`
  Creates the screen of a level within the placement mode. This is the screen which is presented to the user upon entering a recoloring or termedit level.

  Parameters

`controller`	the game controller, which is responsible for the played level

6.15.9 `public class ProfileSetAvatarScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen which is used for both creating a new account with a given avatar as well as changing the avatar of an existing account. For reference see "Pflichtenheft 10.5.14 / Abbildung 23".

Constructors

- `public ProfileSetAvatarScreen(ProfileController controller)`
  Creates the screen that is shown to the player while changing his player avatar.

  Parameters

`controller`	the profile controller, which is responsible for the currently selected profile

6.15.10 `public class ProfileSetNameScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen which is used for both creating a new account with a given name as well as changing the name of an existing account. For reference see "Pflichtenheft 10.5.13 / Abbildung 22".

Constructors

- `public` **`ProfileSetNameScreen`**`(ProfileController controller)`
  Creates the screen that is shown to the player while changing his player name.

  Parameters

`controller`	the profile controller, which is responsible for the currently selected profile

6.15.11 `public class SelectProfileScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen within which the player can change the name of his profile. For reference see "Pflichtenheft 10.5.12 / Abbildung 21".

Constructors

- `public` **`SelectProfileScreen`**`(ProfileController controller)`
  Creates the screen that is shown to the player while changing his profile.

  Parameters

`controller`	the profile controller, which is responsible for the profiles

6.15.12 `public class SettingsScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen within which the player can see the chosen settings and change dem according to his will. For reference see “Pflichtenheft 10.5.10 / Abbildung 19”.

Constructors

- `public SettingsScreen(SettingController controller)`
  Creates the screen that is shown to the player while changing his profile’s settings.

  Parameters

`controller`	the settings controller, which is responsible for the currently selected profile

6.15.13 `public class SimulationModeScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen which is shown during the evaluation-phase of a level. For reference see “Pflichtenheft 10.5.5 / Abbildung 14”.

Constructors

- `public SimulationModeScreen(GameController controller)`
  Creates the screen of a level within the simulation mode. This is the screen which is presented to the user upon pressing the “start simulation button“ within the placement mode screen within a recoloring or termedit level.

  Parameters

`controller`	the game controller, which is responsible for the played level

6.15.14 `public class SplashScreen extends de.croggle.ui.screens.AbstractScreen`

Description

The screen that is shown when the app is started freshly. Will show the app logo and name or something.

Constructors

- `public` **`SplashScreen()`**
  Creates the empty splash screen.

6.15.15 `public class StatisticScreen extends de.croggle.ui.screens.AbstractScreen`

Description

Screen which enables the teacher or parent to control the progress of every user. For reference see “Pflichtenheft 10.5.11 / Abbildung 20”.

Constructors

- `public` **`StatisticScreen`**`(StatisticController controller)`
  Creates the screen within which a parent or teacher can control the player’s progress and statistics.

  Parameters

`controller`	the statistic controller, which is responsible for the statistics

6.16 Package `de.croggle.util`

6.16.1 `public class RingBuffer<T>`

Description

Utility construct, which simplifies saving the last 30 steps immensely.

Constructors

- `public` **`RingBuffer`**`(int size)`
  Creates a ringbuffer with a given size.

  Parameters

  `size` the size with which the ringbuffer will be created. 30 in our case.

Methods

- `public T` **`pop`**`()`
  Removes the topmost object of the ringbuffer and returns it.

  Returns

  the object which used to be on top of the ringbuffer.

- `public void` **`push`**`(T obj)`
  Places the given object on top of the ringbuffer.

  Parameters

  `obj` the object which will be placed upon the ringbuffer.

7 Eigene Exceptions

ColorOverflowException Wird geworfen, falls die darzustellende Konstellation mehr als 30 Farben benötigen würde. Kann bei Alpha-Konversion auftreten. Level sollten so entworfen werden, dass dieser Fall bei einer üblichen Lösung nicht eintritt.

AlligatorOverflowException Wird geworfen, falls die zulässige Maximalanzahl an InternalBoardObjects in einem Board von 300 überschritten wird. Kann beim Hinzufügen von (alten) Alligatoren und Eiern zur Konstellation auf dem Spielfeld auftreten (v.a. im Sandbox-Modus). Level sollten so entworfen werden, dass dieser Fall bei einer üblichen Lösung nicht eintritt.

ProfileOverflowExeption Wird geworfen, falls mehr als die 6 möglichen Profile auftreten. UI sollte so entworfen werden, dass wenn bereits 6 Profile existieren die "Neues Profil Erstellen"Funktionalität nicht mehr zugänglich ist.

IllegalBoardException Wird geworfen, falls nach Start der Simulation festgestellt wird, dass ein nicht gültiger Lambda-Term vorliegt.

8 Anhang

8.1 Klassendiagramm

Im Klassendiagramm ist die komplette Klassenstruktur der Applikation „Croggle“ dargestellt. Eine ausführliche Beschreibung der Klassen befindet sich in diesem Dokument unter dem Abschnitt „Java Klassendokumentation“.

8.2 Sequenzdiagramme

8.2.1 Profilerstellung

Im Sequenzdiagramm zur Profilerstellung wird die Abfolge der Methodenaufrufe bei der Neuerstellung eines Nutzerprofils dargestellt.

8.2.2 Profilwechsel

Dieses Sequenzdiagramm beschreibt die Lade- und Speichervorgänge und die damit verbundenen Methodenaufrufe beim Wechseln eines Profils.

8.2.3 β-Reduktion

In diesem Sequenzdiagramm wird der Ablauf einer einfachen β-Reduktion ohne α-Konversion am Beispiel des Lambda-Terms $(\lambda x.x)y$ dargestellt.

8.2.4 α-Konversion

Hier wird die Abfolge der Methodenaufrufe bei der α-Konversion, die im vorherigen Sequenzdiagram ausgelassen wurde, allgemein und ausführlich gezeigt.

8.2.5 Rendering

In diesem UML-Diagramm wird der Ablauf der Methodenaufrufe beim Rendering von Elementen gezeigt.

8.2.6 Appstart

Hier werden die Initialisierungsreihenfolge und sonstige Vorgänge bei einem Start der App deutlich gemacht.