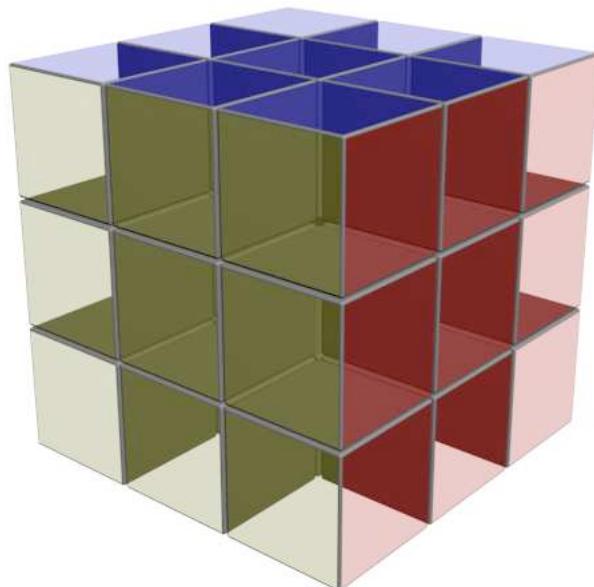




»Entwurf« v 1.0

YaRC

Yet another Rubik Cube



Phase	Phasenverantwortliche(r)	E-Mail

Inhaltsverzeichnis

1 Einleitung	3
2 Aufbau	4
2.1 Architektur	4
2.2 Klassendiagramm	5
3 Klassenbeschreibung	6
3.1 Übersicht	6
3.2 Model	7
3.3 Controller	15
3.4 View	23
3.4.1 2D View	23
3.4.2 3D View	30
4 Abläufe	37
4.1 Spielstart	37
4.2 Programmstart und Änderung des Schwierigkeitsgrades	38
4.3 Drehung	38
4.4 Hilfefunktion	39
4.5 Pause durch Anruf	39
4.6 Pause durch Benutzer	40
4.7 Spiel gelöst	41
4.8 Speichern und beenden	42
5 Entwurfdaten	43
5.1 Ressourcen-Verzeichnis	43
5.2 Dateneinträge	43
6 Klassenindex	44
7 Anhang	45
Glossar	58

1 Einleitung

Mit diesem Entwurfsdokument wollen wir die Systemarchitektur unseres Handyspiels "YaRC" präsentieren. Wir stellen das Klassendiagramm vor, das gemäß dem MVC-Prinzip aufgebaut ist. Die einzelnen Klassen werden danach genauer beschrieben. Außerdem sind mehrere Sequenzdiagramme vorhanden, die den zeitlichen Ablauf verschiedener typischer Vorgänge zeigen.

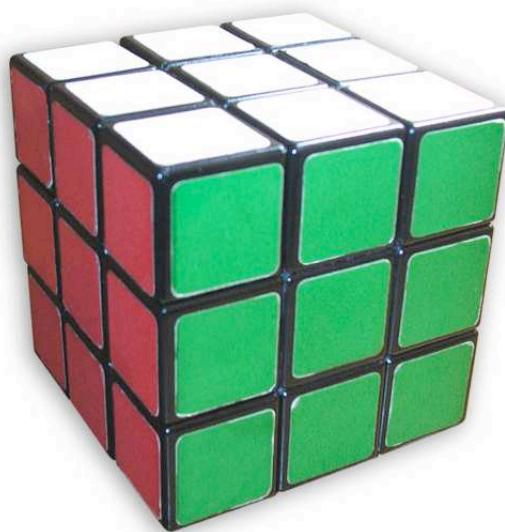
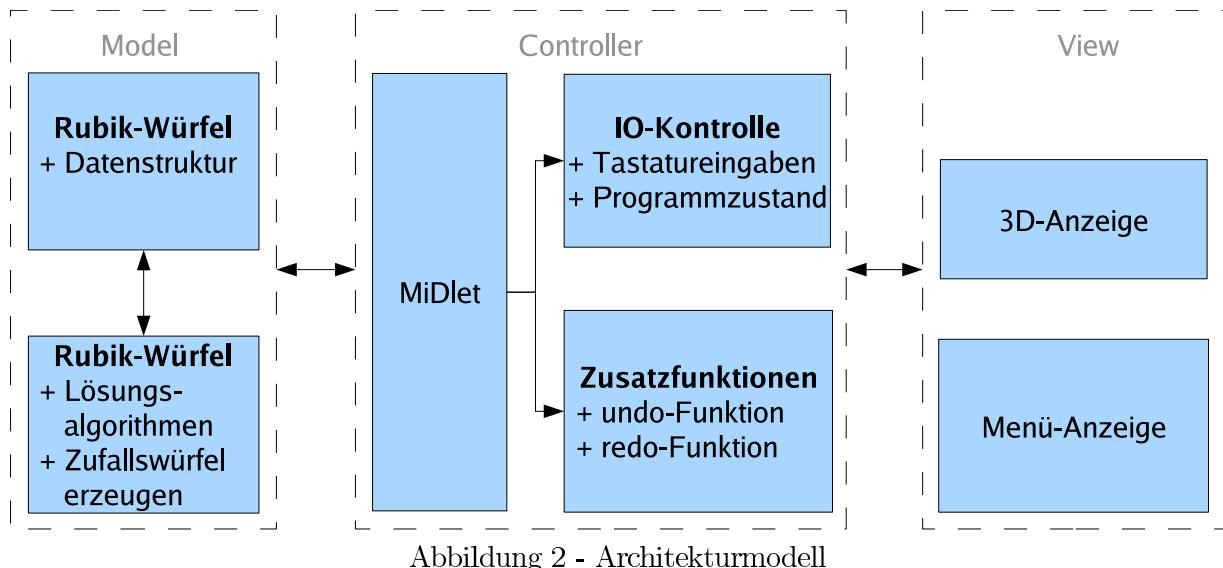


Abbildung 1 - Rubik-Würfel

2 Aufbau

2.1 Architektur



Die Grundstruktur dieses Programms basiert auf der [MVC](#)-Architektur. Dabei soll die Dreiteilung in Modell, View und Controller sowohl eine erleichterte Modifikation am Programmcode, als auch die Portierung auf andere Plattformen ermöglichen. Die drei Komponenten sollen dabei folgende Aufgaben erfüllen:

Model

Das Model enthält sowohl die Datenstruktur des Rubikwürfels, als auch die Algorithmen, die auf den Würfel angewandt werden.

Die Datenstruktur speichert die Ausgangsposition der Teilwürfel (aus denen der Rubikwürfel aufgebaut ist) und den aktuellen Zustand des Rubikwürfels.

An Algorithmen werden zum einen die Lösungsalgorithmen zum kompletten und schrittweisen Lösen des Würfels benötigt, zum anderen eine Funktion für das Erstellen eines zufällig verdrehten Würfels.

Controller

Der Controller ist die Zentraleinheit des Systems. Er koordiniert den Start, das Unterbrechen und das Beenden des Handyspiels. Ebenfalls koordiniert er die Tastatureingaben, die abhängig vom jeweiligen Programmzustand eine entsprechende Reaktion im Model bzw. View bewirken. Zuletzt übernimmt der Controller auch die Zusatzfunktionen wie z.B. die [undo](#) - und [redo](#) - Funktionen, oder eventuelle Wunschfunktionen (z.B. Tonwiedergabe).

View

Die View stellt dem Programm die Ausgabe auf dem Bildschirm ([LCD](#) des Handys) dar. Dabei gibt es die zweidimensionale Darstellung der Menüs und die dreidimensionale Anzeige des Würfels im Spielmodus.

2.2 Klassendiagramm

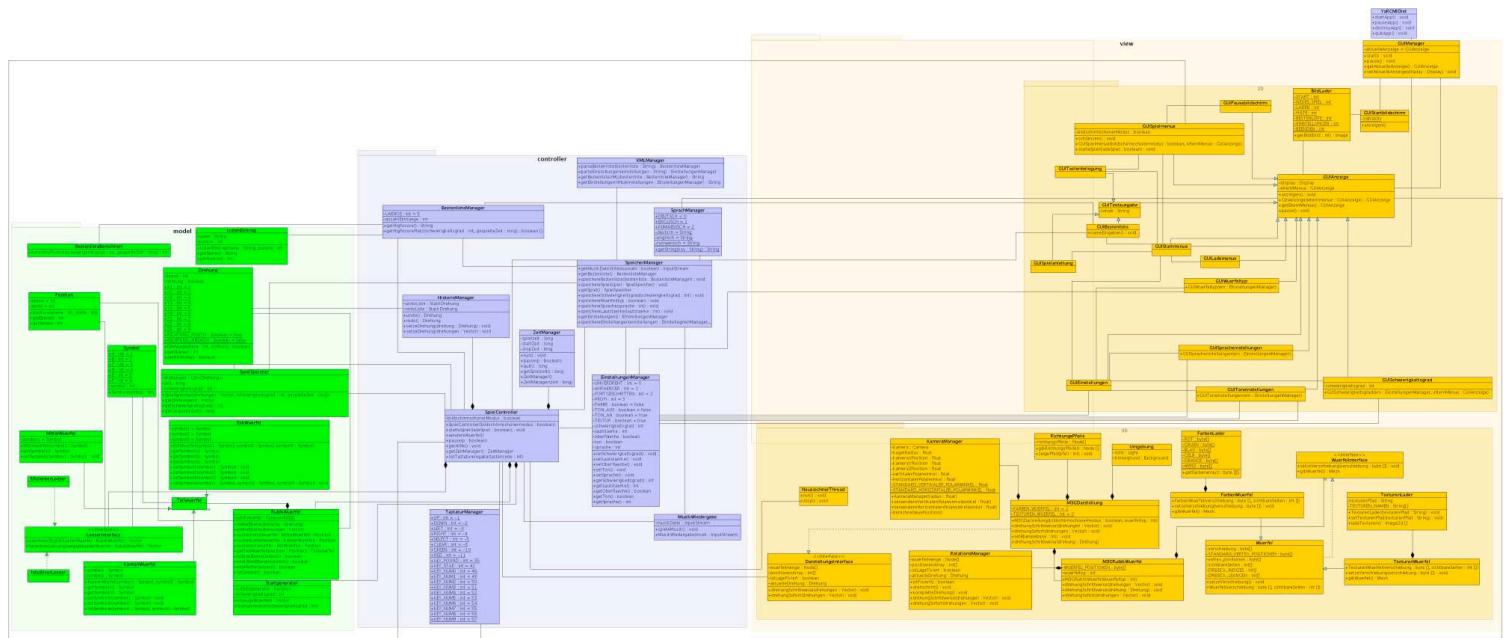


Abbildung 3 - Klassendiagramm (klein)

1 auf 12- Ansicht: siehe Anhang!

3 Klassenbeschreibung

3.1 Übersicht

Model:	Seite	2D View:	Seite
RubikWuerfel	8	GUIStartbildschirm	24
Teilwuerfel	8	GUIManager	24
KantenWuerfel	9	GUIAnzeige	25
EckWuerfel	9	GUIStartmenue	25
MittelWuerfel	10	GUISpielmenue	25
LoeserInterface	10	GUILademenu	26
EffizienterLoeser	10	GUITextausgabe	26
IntuitiverLoeser	11	GUIBestenliste	26
Startgenerator	11	GUISpielanleitung	27
BestenlisteBerechnen	11	GUITastenbelegung	27
ListenEintrag	11	GUIEinstellungen	27
SpielSpeicher	12	GUISpracheinstellungen	27
Drehung	12	GUISchwierigkeitsgrad	28
Position	13	GUIToneinstellungen	28
Symbol	14	GUITWuerfeltyp	28
		GUIPausebildschirm	28
		BildLader	29

Controller:	Seite	3D View:	Seite
YaRCMIDlet	16	DarstellungsInterface	30
SpielController	16	M3GDarstellung	31
EinstellungenManager	17	M3GRubikWuerfel	31
TastaturManager	18	Richtungspfeile	32
ZeitManager	19	KameraManager	32
HistorieManager	20	Umgebung	33
XMLManager	20	RotationsManager	33
SprachManager	21	WuerfelInterface	34
BestenlisteManager	21	Wuerfel	34
SpeicherManager	22	FarbenWuerfel	34
MusikWiedergabe	22	TexturenWuerfel	35
		FarbenLader	35
		TexturenLader	36
		NeuzeichnerThread	36

3.2 Model

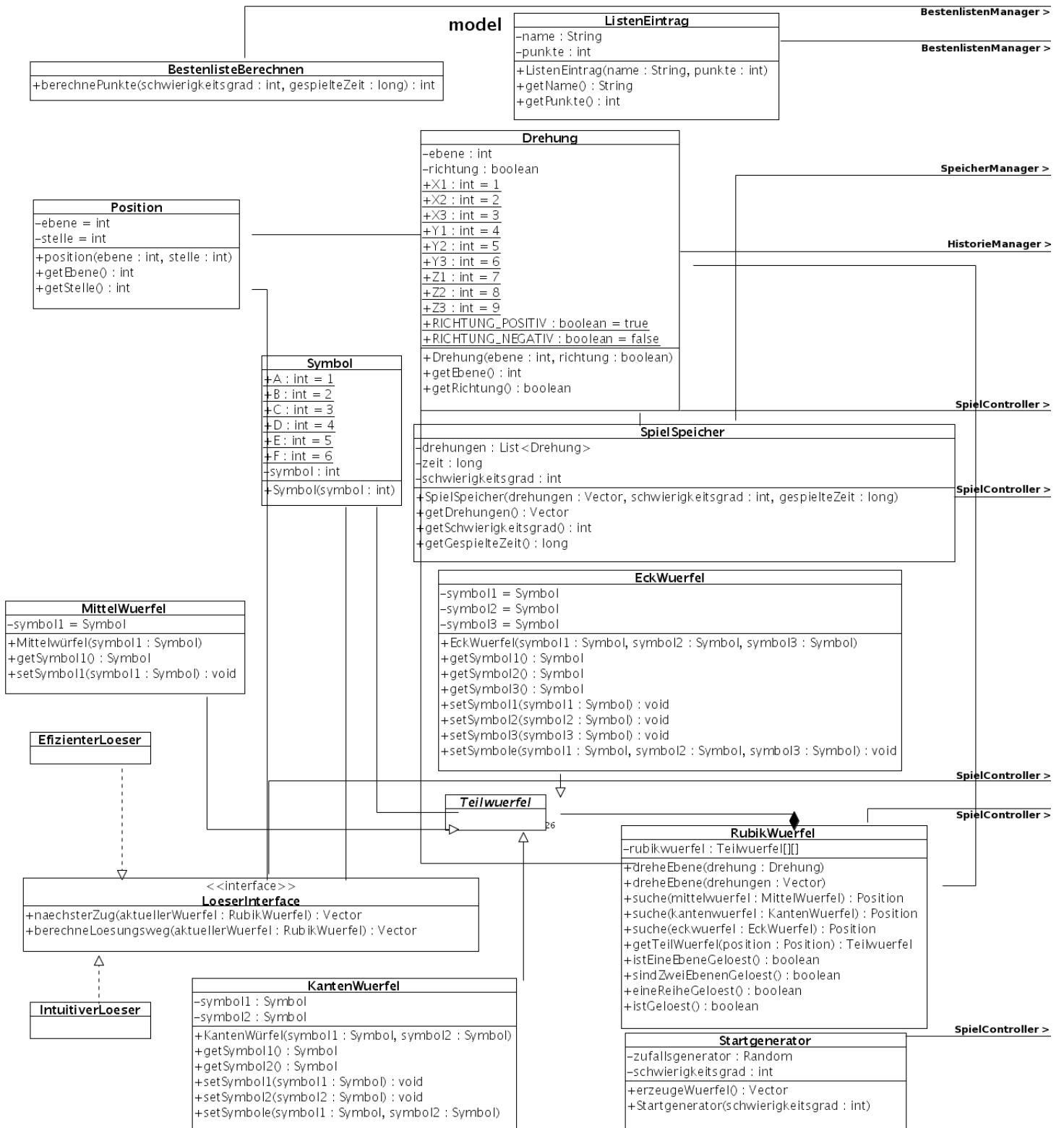


Abbildung 4 - Model

► RubikWuerfel

RubikWuerfel
<ul style="list-style-type: none"> - rubikwuerfel : Teilwuerfel[][] + dreheEbene(Drehung drehung) + dreheEbene(Vector drehungen) + suche(MittelWuerfel mittelwuerfel) : Position + suche(KantenWuerfel kantenwuerfel) : Position + suche(EckWuerfel eckwuerfel) : Position + getTeilwuerfel(Position position) : Teilwuerfel + istGeloest() : boolean + sindZweiEbenenGeloest() : boolean + eineReiheGeloest() : boolean + istGeloest() : boolean

Attribute

- *private Teilwuerfel[][] rubikwuerfel* Das Attribut rubikwuerfel ist eine 9x3-Matrix, die alle 26 Teilwuerfel enthält. Der unsichtbare mittlere Würfel wird mit null referenziert.

Methoden

- *public dreheEbene(Drehung drehung)* Führt intern in der Datenstruktur eine Drehung aus.
- *public dreheEbene(Vector drehungen)* Führt intern in der Datenstruktur mehrere Drehungen aus.
- *public Position suche(MittelWuerfel mittelwuerfel)* Diese Methode sucht in rubikwuerfel einen Mittelwuerfel mit entsprechenden Symbol und gibt dessen Position zurück.
- *public Position suche(KantenWuerfel kantenwuerfel)* Diese Methode sucht in rubikwuerfel einen Kantenwuerfel mit den entsprechenden 2 Symbolen und gibt dessen Position zurück.
- *public Position suche(EckWuerfel eckwuerfel)* Diese Methode sucht in rubikwuerfel einen Eckwuerfel mit den entsprechenden 3 Symbolen und gibt dessen Position zurück.
- *public Teilwuerfel getTeilwuerfel(Position position)* Es wird der Teilwuerfel zurückgegeben, der sich an der Position position befindet.
- *public boolean istEineEbeneGeloest()* Diese Methode gibt true zurück, wenn bereits eine Ebene des rubikwuerfels gelöst ist, d.h. es gibt eine Seite bei der alle Farben gleich sind und zusätzlich jeweils an den angrenzenden Seiten die angrenzende Reihe gleiche Farben hat.
- *public boolean sindZweiEbenenGeloest()* Diese Methode gibt true zurück, falls bereits 2 Ebenen gelöst sind, d.h. eine Ebene ist gelöst und zusätzlich stimmt jeweils an den angrenzenden Seiten die 2.Reihe mit der ersten Reihe dieser Seite überein.
- *public boolean eineReiheGeloest()* Diese Methode gibt true zurück, wenn bereits eine Reihe gelöst ist, d.h. es gibt eine Reihe mit jeweils gleichen Farben auf beiden Seiten.
- *public boolean istGeloest()* Diese Methode gibt true zurück, falls der Rubik-Würfel komplett gelöst ist, sonst false.

► Teilwuerfel

Teilwuerfel

Die abstrakte Klasse Teilwuerfel implementiert die Datenstruktur der Teilwürfel, aus denen ein RubikWuerfel zusammengesetzt ist.

Attribute

Methoden

► KantenWuerfel

KantenWuerfel
- symbol1 : Symbol
- symbol2 : Symbol
+ KantenWuerfel(Symbol symbol1, Symbol symbol2)
+ getSymbol1() : Symbol
+ getSymbol2() : Symbol
+ setSymbol1(Symbol symbol1)
+ setSymbol2(Symbol symbol2)
+ setSymbole(Symbol symbol1, Symbol symbol2)

Die Klasse KantenWuerfel enthält 2 sichtbare Seiten. Vorderseite = 1.Seite, obere Seite = 2.Seite.

Attribute

- *private Symbol symbol1* Dieses Symbol symbol1 befindet sich auf der Vorderseite des Kantenwürfels.
- *private Symbol symbol2* Dieses Symbol symbol2 befindet sich auf der oberen Seite des Kantenwürfels.

Methoden

- *public KantenWuerfel(Symbol symbol1, Symbol symbol2)* Erzeugt ein neues KantenWuerfel-Objekt.
- *public Symbol getSymbol1()* Diese Methode gibt das Symbol, das sich auf der Vorderseite befindet, zurück.
- *public Symbol getSymbol2()* Diese Methode gibt das Symbol, das sich auf der oberen Seite befindet, zurück.
- *public void setSymbol1(Symbol symbol1)* Diese Methode setzt das Symbol, das sich auf der Vorderseite befindet, fest.
- *public void setSymbol2(Symbol symbol2)* Diese Methode setzt das Symbol, das sich auf der oberen Seite befindet, fest.
- *public void setSymbole(Symbol symbol1, Symbol symbol2)* Diese Methode setzt die beiden Symbole des Kantenwürfels fest.

► EckWuerfel

EckWuerfel
- symbol1 : Symbol
- symbol2 : Symbol
- symbol3 : Symbol
+ EckWuerfel(Symbol symbol1, Symbol symbol2, Symbol symbol3)
+ getSymbol1() : Symbol
+ getSymbol2() : Symbol
+ getSymbol3() : Symbol
+ setSymbol1(Symbol symbol1)
+ setSymbol2(Symbol symbol2)
+ setSymbol3(Symbol symbol3)
+ setSymbole(Symbol symbol1, Symbol symbol2, Symbol symbol3)

Die Klasse EckWuerfel enthält 3 sichtbare Seiten. Vorderseite = 1.Seite, obere Seite = 2.Seite, linke Seite = 3.Seite.

Attribute

- *private Symbol symbol1* Dieses Symbol symbol1 befindet sich auf der Vorderseite des Eckwürfels.
- *private Symbol symbol2* Dieses Symbol symbol2 befindet sich auf der oberen Seite des Eckwürfels.
- *private Symbol symbol3* Dieses Symbol symbol3 befindet sich auf der linken Seite des Eckwürfels.

Methoden

- *public EckWuerfel(Symbol symbol1, Symbol symbol2, Symbol symbol3)* Erzeugt ein neues EckWuerfel-Objekt.
- *public Symbol getSymbol1()* Diese Methode gibt das Symbol, das sich auf der Vorderseite befindet, zurück.
- *public Symbol getSymbol2()* Diese Methode gibt das Symbol, das sich auf der oberen Seite befindet, zurück.
- *public Symbol getSymbol3()* Diese Methode gibt das Symbol, das sich auf der linken Seite befindet, zurück.

3 Klassenbeschreibung

- *public void setSymbol1(Symbol symbol1)* Diese Methode setzt das Symbol, das sich auf der Vorderseite befindet, fest.
- *public void setSymbol2(Symbol symbol2)* Diese Methode setzt das Symbol, das sich auf der oberen Seite befindet, fest.
- *public void setSymbol3(Symbol symbol3)* Diese Methode setzt das Symbol, das sich auf der linken Seite befindet, fest.
- *public void setSymbole(Symbol symbol1, Symbol symbol2, Symbol symbol3)* Diese Methode setzt die drei Symbole eines Eckwürfels fest.

► MittelWuerfel

MittelWuerfel
- symbol1 : Symbol
+ MittelWuerfel(Symbol symbol1)
+ getSymbol1() : Symbol
+ setSymbol1(Symbol symbol1)

Die Klasse MittelWuerfel enthält ein Symbol, welches auf seiner Vorderseite sichtbar ist.

Attribute

- *private Symbol symbol1* Dieses Symbol symbol1 befindet sich auf der Vorderseite des Mittelwürfels.

Methoden

- *public MittelWuerfel(Symbol symbol1)* Erzeugt ein neues MittelWuerfel-Objekt.
- *public Symbol getSymbol1()* Diese Methode gibt das Symbol, das sich auf der Vorderseite befindet, zurück.
- *public void setSymbol1(Symbol symbol1)* Diese Methode setzt das Symbol, das sich auf der Vorderseite befindet, fest.

► LoeserInterface

LoeserInterface
+ Vector naechsterZug (RubikWuerfel aktuellerWuerfel)
+ Vector berechneLoesungsweg (RubikWuerfel aktuellerWuerfel)

LoeserInterface ist die Schnittstelle für die Lösungsalgorithmen und legt somit die Anforderungen an den Löser fest.

Attribute

Methoden

- *public Vector naechsterZug (RubikWuerfel aktuellerWuerfel)* Gibt die Drehungen für den nächsten Zug zurück.
- *public Vector berechneLoesungsweg (RubikWuerfel aktuellerWuerfel)* Gibt die fürs Lösen des Würfels nötigen Schritte zurück.

► EffizienterLoeser

EffizienterLoeser

Implementiert LoeserInterface mit einem effizienten Lösungsalgorithmus.

Attribute

Methoden

► IntuitiverLoeser

IntuitiverLoeser

Implementiert LoeserInterface mit einem intuitiven Lösungsalgorithmus.

Attribute

Methoden

► Startgenerator

Startgenerator
- zufallsgenerator : Random
- schwierigkeitsgrad : int
+ Startgenerator(int schwierigkeitsgrad)
+ erzeugeWuerfel(int Schwierigkeitsgrad) : Vector

Die Klasse Startgenerator ist zuständig für das Erstellen eines zufällig verdrehten Würfels. Sie wird vom SpielController aus aufgerufen.

Attribute

- *private Random zufallsgenerator* Es werden Zufallszahlen erzeugt.
- *private int schwierigkeitsgrad* Der aktuelle Schwierigkeitsgrad des Startgenerators.

Methoden

- *public Startgenerator(int schwierigkeitsgrad)* Konstruktor mit Schwierigkeitsgrad als Eingabe.
- *public Vector erzeugeWuerfel()* Gibt die Liste von Drehungen zurück, die zum Erstellen eines zufälligen Würfels mit entsprechendem Schwierigkeitsgrad verwendet werden.

► BestenlisteBerechnen

BestenlisteBerechnen
berechnePunkte(int schwierigkeitsgrad, long gespielteZeit) : int

Die Klasse BestenlisteBerechnen ist zuständig für das Berechnen der erreichten Punkte.

Attribute

Methoden

- *public int berechnePunkte(int schwierigkeitsgrad, long gespielteZeit)* Punkte werden abhängig vom Schwierigkeitsgrad und der gespielten Zeit berechnet.

► ListenEintrag

ListenEintrag
- name : String
- punkte : int
+ ListenEintrag(String name, int punkte)
+ getName() : String
+ getPunkte() : int

Stellt einen einzelnen Eintrag von der Bestenliste dar.

Attribute

- *private String name* Name der Person, die den Highscore erzielt hat.
- *private int punkte* Erzielter Highscore.

Methoden

- *public ListenEintrag(String name, int punkte)* Konstruktor für einen ListenEintrag, dem der Name des Benutzers und die erzielten Punkte übergeben werden.
- *public String getName()* Methode, die den Namen des Benutzers für den entsprechenden Eintrag zurück gibt.
- *public int getPunkte()* Methode, die die erzielten Punkte für den entsprechenden Eintrag zurück gibt.

► SpielSpeicher

SpielSpeicher
- drehungen : Vector
- zeit : long
- schwierigkeitsgrad : int
+ SpielSpeicher(Vector drehungen, int schwierigkeitsgrad, long gespielteZeit)
+ getDrehungen() : Vector
+ getSchwierigkeitsgrad() : int
+ getGespielteZeit() : long

Die Klasse SpielSpeicher repräsentiert einen Spielzustand.

Attribute

- *private Vector drehungen* Die Drehungen, die zum Erzeugen des Spielzustandes auszuführen sind.
- *private long zeit* Die bisher gespielte Zeit eines gespeicherten Spieles.
- *private int schwierigkeitsgrad* Der Schwierigkeitsgrad eines gespeicherten Spieles.

Methoden

- *public SpielSpeicher(Vector drehungen, int schwierigkeitsgrad, long gespielteZeit)* Konstruktor zum Erzeugen eines SpielSpeicher.
- *public Vector getDrehungen()* Gibt die Drehungen zurück, die zum erzeugen des Spielzustandes auszuführen sind.
- *public int getSchwierigkeitsgrad()* Gibt den Schwierigkeitsgrad eines gespeicherten Spieles zurück.
- *public long getGespielteZeit()* Gibt die bisher benötigte Zeit eines gespeicherten Spieles zurück.

► Drehung

Drehung
- achse : int
- richtung : boolean
+ RICHTUNG_POSITIV : boolean
+ RICHTUNG_NEGATIV : boolean
+ X1 : int
+ X2 : int
+ X3 : int
+ Y1 : int
+ Y2 : int
+ Y3 : int
+ Z1 : int
+ Z2 : int
+ Z3 : int
+ Drehung(int ebene, boolean richtung)
+ getEbene() : int
+ getRichtung() : boolean

Die Klasse Drehung beschreibt ein Drehungsobjekt, welches aus Achse und Richtung besteht.

Attribute

- *private int ebene* Die zu drehende Ebene.
- *private boolean richtung* Die Richtung der Drehung.
- *public static final boolean RICHTUNG_POSITIV = true* Drehung im Uhrzeigersinn.
- *public static final boolean RICHTUNG_NEGATIV = false* Drehung gegen den Uhrzeigersinn.
- *public static final int X1=1* Die Ebene X1. (siehe Abbildung)
- *public static final int X2=2* Die Ebene X2. (siehe Abbildung)
- *public static final int X3=3* Die Ebene X3. (siehe Abbildung)
- *public static final int Y1=4* Die Ebene Y1. (siehe Abbildung)
- *public static final int Y2=5* Die Ebene Y2. (siehe Abbildung)

- *public static final int Y3=6* Die Ebene Y3. (siehe Abbildung)
- *public static final int Z1=7* Die Ebene Z1. (siehe Abbildung)
- *public static final int Z2=8* Die Ebene Z2. (siehe Abbildung)
- *public static final int Z3=9* Die Ebene Z3. (siehe Abbildung)

Methoden

- *public Drehung(int ebene, boolean richtung)* Konstruktor zur Erzeugung eines Drehung-Objekts. Wertebereich der Ebene: $1 \leq \text{ebene} \leq 9$
- *public int getEbene()* Gibt die Ebene zurück.
- *public boolean getRichtung()* Gibt die Richtung zurück.

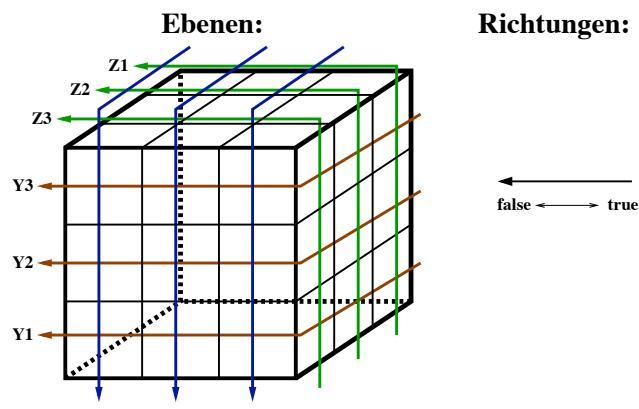


Abbildung 5 - Rotationsebenen eines Rubikwürfels.

► Position

Position
- ebene : int
- stelle : int
+ Position(int ebene, int stelle)
+ getEbene() : int
+ getStelle() : int

Die Klasse Position beschreibt die Lage eines Teilwürfels in einem Rubik-Würfel.

Attribute

- *private int ebene* Die Ebene in der sich der Teilwürfel befindet. Es wird von oben nach unten gezählt. Insgesamt gibt es hier also 3 Ebenen.
- *private int stelle* Gibt die Stelle des Teilwürfels in der Ebene an. Es wird von hinten nach vorne gezählt und in einer Reihe wird von links nach rechts gezählt. Es gibt in einer Ebene 9 Stellen. Außer bei der 2.Ebene ist die 5.Stelle unbesetzt.

Methoden

- *public Position(int ebene, int stelle)* Konstruktor zur Erzeugung eines Position-Objekts
- *public int getEbene()* Methode, die die Nummer der Ebene zurück gibt.
- *public int getStelle()* Methode, die die Stelle in der Ebene zurück gibt.

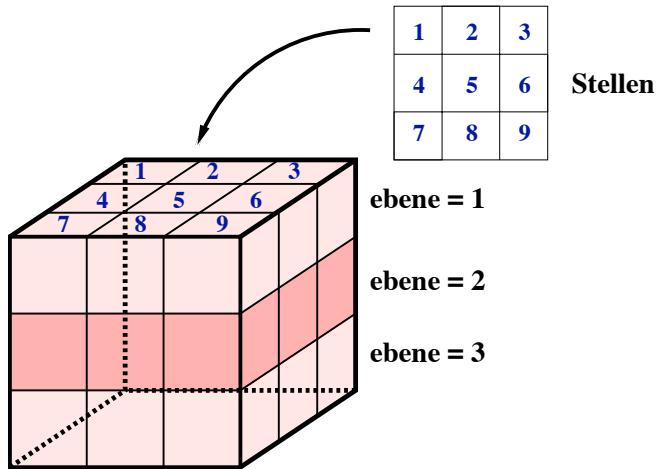


Abbildung 6 - Positionen der Teilwürfel.

► Symbol

Symbol	
-	symbol : int
+	A : int
+	B : int
+	C : int
+	D : int
+	E : int
+	F : int
+	Symbol(int symbol)

Die Klasse Symbol beschreibt ein beliebiges Symbol (z.B. Farbe, Textur). Solche Symbole befinden sich auf den Teilwürfeln.

Attribute

- *private int symbol* Die konkrete Ausprägung des Symbols.
- *public static final int A = 1* Symbol A.
- *public static final int B = 2* Symbol B.
- *public static final int C = 3* Symbol C.
- *public static final int D = 4* Symbol D.
- *public static final int E = 5* Symbol E.
- *public static final int F = 6* Symbol F.

Methoden

- *public Symbol(int symbol)* Konstruktor für die Klasse Symbol. Hier wird ein Integer-Wert übergeben, der die Werte der sechs static-Variablen A - F annehmen kann.

3.3 Controller

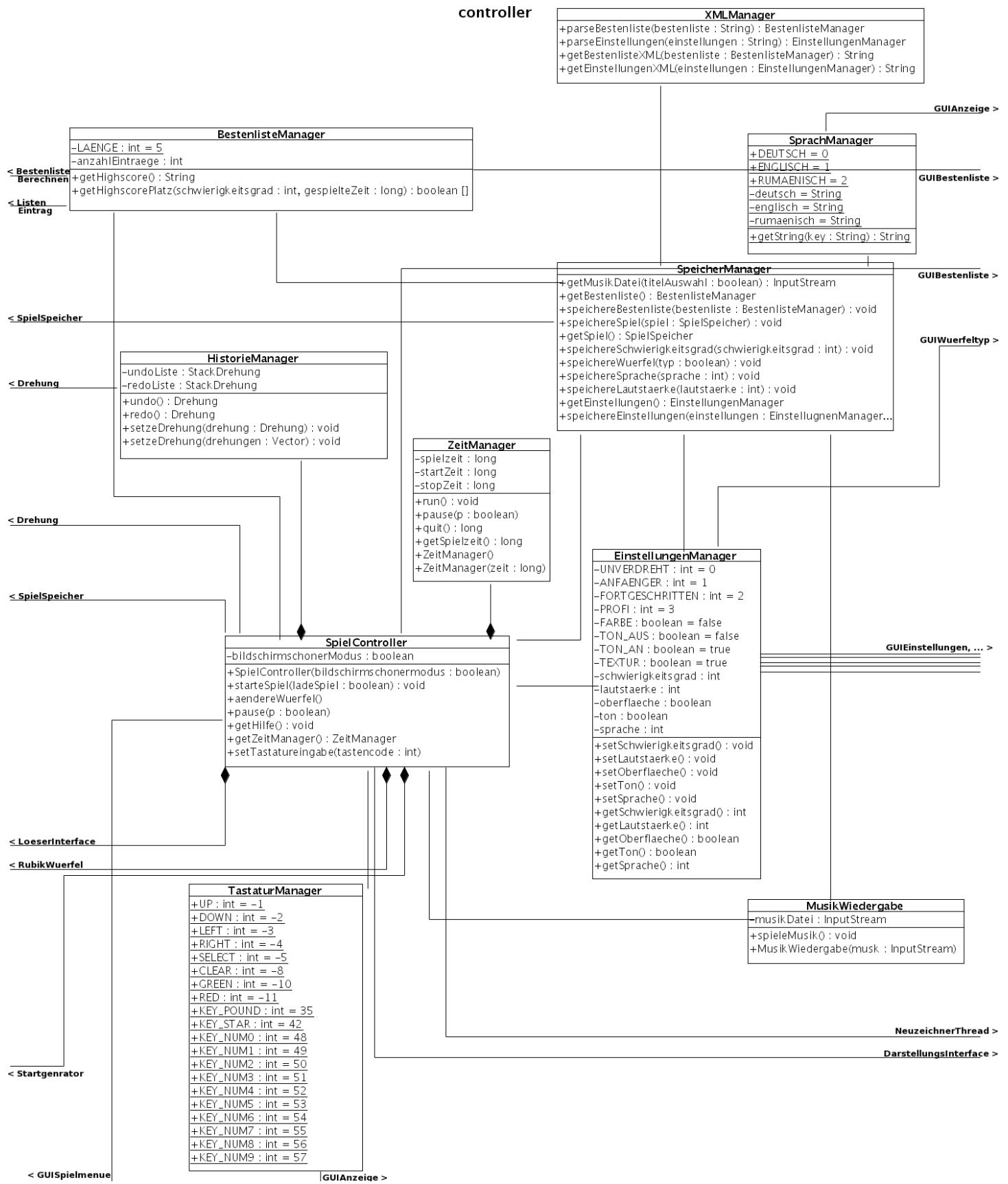


Abbildung 7 - Controller

► YaRCMIDlet

YaRCMIDlet
+startApp() +pauseApp() +destroyApp() +quitApp()

Die Klasse YaRCMIDlet ist zuständig für das Starten, Pausieren und Beenden des MIDlets. Sie ist also der Eintritt in das Programm.

Attribute

Methoden

- *public startApp()* Über diese, von der Klasse MIDlet vorgegebene Methode wird das Programm gestartet.
- *public pauseApp()* Diese Methode wird vom Programmmanager z.B. bei einem eingehenden Anruf aufgerufen.
- *public destroyApp()* In dieser Methode muss das MIDlet "sauber" beendet werden.
- *public quitApp()* Wird vom Programm-Manager zum Beenden des MIDlets aufgerufen.

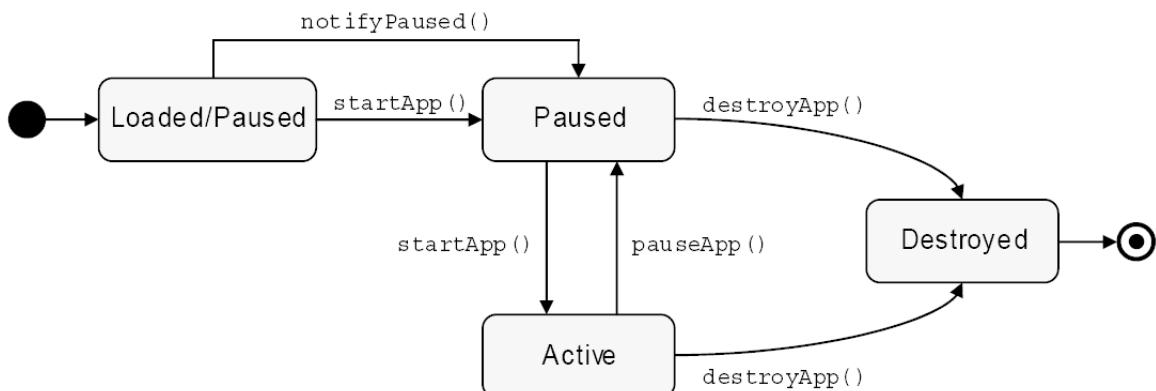


Abbildung 8 - Lebenszyklus eines MIDlets [<http://de.wikipedia.org/wiki/Bild:Lebenszyklus.png>]

► SpielController

SpielController
- bildschirmschonerModus : boolean + SpielController(boolean bildschirmschonermodus) + starteSpiel(boolean ladeSpiel) + aendereWuerfel() + pause(boolean p) + getHilfe() + getZeitManager() : ZeitManager + setTastatureingabe(int tastencode)

Der SpielController übernimmt die Koordination des Spielablaufs. Er veranlasst eine Änderung des Zustands in der Klasse RubikWuerfel und eine entsprechende Reaktion der M3GDarstellung.

Attribute

- *private boolean bildschirmschonerModus* Falls dieser Wert true ist befindet sich das Handyspiel im Bildschirmschonermodus.

Methoden

- *public SpielController(boolean bildschirmschonermodus)* Erzeugt ein neues SpielController- Objekt. Falls der übergebene boolean-Wert true ist wird das Spiel im Bilschirmschonermodus gestartet.
- *public void starteSpiel(boolean ladeSpiel)* Falls der übergebene boolean-Wert true ist, wird das gespeicherte Spiel geladen. Falls kein Spiel gespeichert wurde, oder der boolean-Wert false ist wird ein neues und zufälliges Spiel erzeugt.
- *public void aendereWuerfel()* Veranlasst die Änderung des Würfels in RubikWuerfel und in M3GDarstellung

- *public pause(boolean p)* Startet oder beendet den Pause-Modus, abhängig von p (true = Start, false = Ende).
- *public void getHilfe()* Veranlasst das LoeserInterface die nächsten Lösungsschritte zu berechnen, woraufhin diese Drehungen ausgeführt werden. Diese Methode wird vom GUI Spielmenue aus aufgerufen.
- *public ZeitManager getZeitManager()* Falls die M3GDarstellung den Zeitmanager benötigt (im Spielmodus) kann sie ihn damit abrufen.
- *public void setTastatureingabe(int tastencode)* Diese Methode nimmt die Tasteneigaben der M3GDarstellung entgegen.

► EinstellungenManager

EinstellungenManager	
+ UNVERDREHT : int + ANFAENGER : int + FORTGESCHRITTEN : int + PROFI : int + FARBE : boolean + TEXTUR : boolean + TON_AUS : boolean + TON_AN : boolean - schwierigkeitsgrad : int - lautstaerke : int - oberflaeche : boolean - ton : boolean - sprache : int + setSchwierigkeitsgrad(int schwierigkeitsgrad) : void + setLautstaerke(int lautstaerke) : void + setOberflaeche(boolean oberflaeche) : void + setTon(boolean ton) : void + setSprache(int sprache) : void + getSchwierigkeitsgrad() : int + getLautstaerke() : int + getOberflaeche() : boolean + getTon() : boolean + getSprache() : int	

Die Klasse EinstellungenManager repräsentiert und verwaltet die Einstellungen des Benutzers.

Attribute

- *public static final int UNVERDREHT = 0*
- *public static final int ANFAENGER = 1*
- *public static final int FORTGESCHRITTEN = 2*
- *public static final int PROFI = 3*
- *public static final boolean FARBE = false*
- *public static final boolean TEXTUR = true*
- *public static final boolean TON_AUS = false*
- *public static final boolean TON_AN = true*
- *private int schwierigkeitsgrad* Der aktuell eingestellte Schwierigkeitsgrad.
- *private int lautstaerke* Die aktuell eingestellte Lautstärke.
- *private boolean oberflaeche* Die aktuell eingestellte Oberfläche.
- *private boolean ton* Der aktuell eingestellte Tonmodus (an/aus).
- *private int sprache* Die aktuell eingestellte Sprache.

Methoden

- *public void setSchwierigkeitsgrad(int schwierigkeitsgrad)* Setzt den Schwierigkeitsgrad des Spiels fest.
- *public void setLautstaerke(int lautstaerke)* Setzt die Lautstärke fest.
- *public void setOberflaeche(boolean oberflaeche)* Setzt die Oberfläche des Würfels fest.
- *public void setTon(boolean ton)* Deaktiviert bzw. aktiviert den Ton.
- *public void setSprache(int sprache)* Legt die Sprache fest.
- *public int getSchwierigkeitsgrad()* Liefert den aktuellen Schwierigkeitsgrad.
- *public int getLautstaerke()* Liefert die aktuelle Lautstärke.
- *public boolean getOberflaeche()* Liefert die aktuelle Oberfläche des Würfels.
- *public boolean getTon()* Liefert den aktuellen Tonmodus.
- *public int getSprache()* Liefert die aktuelle Sprache.

► TastaturManager

TastaturManager
+ UP : int
+ DOWN : int
+ LEFT : int
+ RIGHT : int
+ SELECT : int
+ CLEAR : int
+ GREEN : int
+ RED : int
+ KEY_POUND : int
+ KEY_STAR : int
+ KEY_NUM0 : int
+ KEY_NUM1 : int
+ KEY_NUM2 : int
+ KEY_NUM3 : int
+ KEY_NUM4 : int
+ KEY_NUM5 : int
+ KEY_NUM6 : int
+ KEY_NUM7 : int
+ KEY_NUM8 : int
+ KEY_NUM9 : int

Die Klasse TastaturManager definiert die Bezeichnungen der Tasten je nach (von der Canvas) abgefangenem Integer-Wert. Diese Werte unterscheiden sich von denen in der Dokumentation der Canvas-Klasse angegebenen und sind durch einen Testprototypen ermittelt.

Attribute

- *public static final int UP = -1*
- *public static final int DOWN = -2*
- *public static final int LEFT = -3*
- *public static final int RIGHT = -4*
- *public static final int SELECT = -5*
- *public static final int CLEAR = -8*
- *public static final int GREEN = -10*
- *public static final int RED = -11*
- *public static final int KEY_POUND = 35*
- *public static final int KEY_STAR = 42*
- *public static final int KEY_NUM0 = 48*

- *public static final int KEY_NUM1 = 49*
- *public static final int KEY_NUM2 = 50*
- *public static final int KEY_NUM3 = 51*
- *public static final int KEY_NUM4 = 52*
- *public static final int KEY_NUM5 = 53*
- *public static final int KEY_NUM6 = 54*
- *public static final int KEY_NUM7 = 55*
- *public static final int KEY_NUM8 = 56*
- *public static final int KEY_NUM9 = 57*

Methoden

► ZeitManager

ZeitManager
- spielzeit : long - startZeit : long - stopZeit : long + ZeitManager() + ZeitManager(long zeit) + start() + pause(boolean p) + quit() : long + getZeit() : long

Die Klasse ZeitManager realisiert die Zeitmessung während des Spiels. Die Zeitmessung wird vom SpielController aus gestartet, beendet und pausiert. Nach Beenden des Spiels wird die benötigte Zeit dem SpielController übergeben.

Attribute

- *private long spielzeit* Variable für die gemessene Zeit
- *private long startZeit* Variable, die den Startzeitpunkt (Systemzeit) speichert.
- *private long stopZeit* Variable, die den Stopzeitpunkt (Systemzeit) speichert.

Methoden

- *public ZeitManager()* Standardkonstruktor. Wird verwendet, wenn ein neues Spiel gestartet wird und die Zeitmessung von 0 beginnt.
- *public ZeitManager(long zeit)* Konstruktor, der verwendet wird wenn ein gespeichertes Spiel geladen wird. Die Zeitmessung wird von der übergebenen Zeit aus gestartet.
- *public run()* Startet die Zeitmessung.
- *public pause(boolean p)* Startet oder beendet die Zeitmessung, abhängig von p (true = Start, false = Ende).
- *public long quit()* Beendet die Zeitmessung und liefert die Zeit zurück.
- *public long getZeit()* Liefert die aktuell benötigte Zeit.

► HistorieManager

HistorieManager
- undoListe : Stack
- redoListe : Stack
+ undo() : Drehung
+ redo() : Drehung
+ setzeDrehung(Drehung drehung)
+ setzeDrehung(Vector drehungen)

Die Klasse HistorieManager ist zuständig für die Speicherung der Spielzüge und die Realisierung der undo- bzw. redo-Funktionen. Die Spielzüge erhält der HistorieManager vom SpielController.

Attribute

- *private Stack undoListe* Der undo-Stack.
- *private Stack redoListe* Der redo-Stack.

Methoden

- *public Drehung undo()* Gibt die letzte Drehung zurück und legt sie auf den Stack redoListe.
- *public Drehung redo()* Gibt die letzte rückgängig gemachte Drehung zurück und legt sie auf den Stack undoListe.
- *public setzeDrehung(Drehung drehung)* Legt eine gemachte Drehung auf den Stack undoListe.
- *public setzeDrehung(Vector drehungen)* Legt die gemachten Drehungen auf den Stack undoListe.

► XMLManager

XMLManager
+ parseBestenliste(String bestenliste) : BestenlisteManager
+ parseEinstellungen(String einstellungen) : EinstellungenManager
+ getBestenlisteXML(BestenlisteManager bestenliste) : String
+ getEinstellungenXML(EinstellungenManager einstellungen) : String

Die Klasse XMLManager ist zuständig fÃ¼r das Lesen und Schreiben von XML-Dateien und wird ausschließlich von der Klasse SpeicherManager benutzt.

Attribute

Methoden

- *public BestenlisteManager parseBestenliste(String bestenliste)* Wandelt den übergebenen String in ein Objekt der Klasse BestenlisteManager um.
- *public EinstellungenManager parseEinstellungen(String einstellungen)* Wandelt den übergebenen String in ein Objekt der Klasse EinstellungenManager um.
- *public String getBestenlisteXML(BestenlisteManager bestenliste)* Wandelt das übergebene Objekt in einen XML-String um.
- *public String getEinstellungenXML(EinstellungenManager einstellungen)* Wandelt das übergebene Objekt in einen XML-String um.

► SprachManager

SprachManager
+ DEUTSCH : int
+ ENGLISCH : int
+ RUMAENISCH : int
- deutsch : String
- englisch : String
- rumaenisch : String
+ getString(String key) : String

Die Klasse SprachManager ist zuständig für die Realisierung der möglichen Spracheinstellungen. Er fordert die entsprechende SprachDatei vom SpeicherManager und übergibt sie der GUIAnzeige.

Attribute

- *public static final int DEUTSCH = 0*
- *public static final int ENGLISCH = 1*
- *public static final int RUMAENISCH = 2*
- *private static String deutsch* Deutsch
- *private static String englisch* Englisch
- *private static String rumaenisch* Rumaenisch

Methoden

- *public static String getString(String key)* Liefert die Bezeichnung in der eingestellten Sprache zurück.

► BestenlisteManager

BestenlisteManager
- LAENGE : int
- anzahlEintraege : int
- punkte : int
- bestenliste ListenEintrag[]
+ getHighscore() : String
+ getHighscorePlatz(int schwierigkeitsgrad, long gespielteZeit) : boolean[]
+ neuerName(String name)
+ getBestenliste() : ListenEintrag[]

Die Klasse BestenlisteManager ist zuständig für die Verwaltung der Bestenliste. Der BestenlisteManager erhält die erreichte Punktzahl vom SpielController und prüft, ob diese einen Highscoreplatz bekommen.

Attribute

- *private int LAENGE = 5* Maximale Größe der Bestenliste.
- *private int anzahlEintraege* Anzahl der gespeicherten Einträge.
- *private int punkte* Die zuletzt berechneten Punkte.
- *private ListenEintrag[] bestenliste* Enthält alle Einträge, die in der Bestenliste gespeichert sind. Länge des Arrays = LAENGE, bestenliste[0] = erster Platz in der Highscore, bestenliste[i] = (i+1).ter Platz in der Highscore.

Methoden

- *public String getHighscore()* Gibt die Bestenliste für die Darstellung zurück.
- *public boolean[] getHighscorePlatz(int schwierigkeitsgrad, long gespielteZeit)* Ermittelt, ob die Punkte den Highscore knacken (wird vom ersten Boolean angezeigt) und ob die Punkte einen Eintrag in die Bestenliste erhalten (wird vom zweiten Boolean angezeigt).
- *public void neuerName(String name)* Dieser Name bekommt mit den entsprechenden Punkten einen ListenEintrag.
- *public ListenEintrag[] getBestenliste()* Es wird das Array bestenliste zurückgegeben.

► SpeicherManager

SpeicherManager
<pre>+ getMusikDatei(boolean titelAuswahl) : InputStream + getSprachDatei() + getBestenliste() : BestenlisteManager + speichereBestenliste(BestenlisteManager bestenliste) + speichereSpiel(SpielSpeicher spiel) + getSpiel() : SpielSpeicher + getEinstellungen() : EinstellungenManager + speichereEinstellungen(EinstellungenManager einstellungen) : void + speichereSchwierigkeitsgrad(int schwierigkeitsgrad) : void + speichereWuerfeltyp(boolean typ) : void + speichereSprache(int sprache) : void + speichereLautstaerke(int lautstaerke) : void</pre>

Die Klasse SpeicherManager ist zuständig für den lesenden und schreibenden Zugriff auf den Handyspeicher.

Attribute

Methoden

- *public InputStream getMusikDatei(boolean titelAuswahl)* Liest die Musik-Dateien aus dem Speicher.
- *public getSprachDatei()* Liest die Sprach-Dateien aus dem Speicher.
- *public BestenlisteManager getBestenliste()* Liest die Bestenliste aus dem Speicher.
- *public void speichereBestenliste(BestenlisteManager bestenliste)* Speichert die Bestenliste.
- *public void speichereSpiel(SpielSpeicher spiel)* Speichert ein Spiel.
- *public SpielSpeicher getSpiel()* Liest ein gespeichertes Spiel aus dem Speicher.
- *public void speichereEinstellungen(EinstellungenManager einstellungen)* Speichert die Einstellungen eines Spiels.
- *public void speichereSchwierigkeitsgrad(int schwierigkeitsgrad)* Speichert den eingestellten Schwierigkeitsgrad.
- *public void speichereWuerfeltyp(boolean typ)* Speichert den eingestellten Würfeltyp.
- *public void speichereSprache(int sprache)* Speichert die eingestellte Sprache.
- *public void speichereLautstaerke(int lautstaerke)* Speichert die eingestellte Lautstärke.
- *public EinstellungenManager getEinstellungen()* Liest und liefert die gespeicherten Einstellungen.

► MusikWiedergabe

MusikWiedergabe
<pre>- musikDatei : String - highscoreMusikDatei : String + spieleMusik() + MusikWiedergabe(InputStream musik) + spieleMusik() : void</pre>

Die Klasse MusikWiedergabe ist zuständig für das Abspielen von Musik. Dies wird vom SpielController aus aufgerufen.

Attribute

- *private InputStream musikDatei* MusikDatei für die Melodie.

Methoden

- *public MusikWiedergabe(InputStream musik)*
- *public void spieleMusik()* Spielt die Melodie beim Lösen eines Würfels

3.4 View

3.4.1 2D View

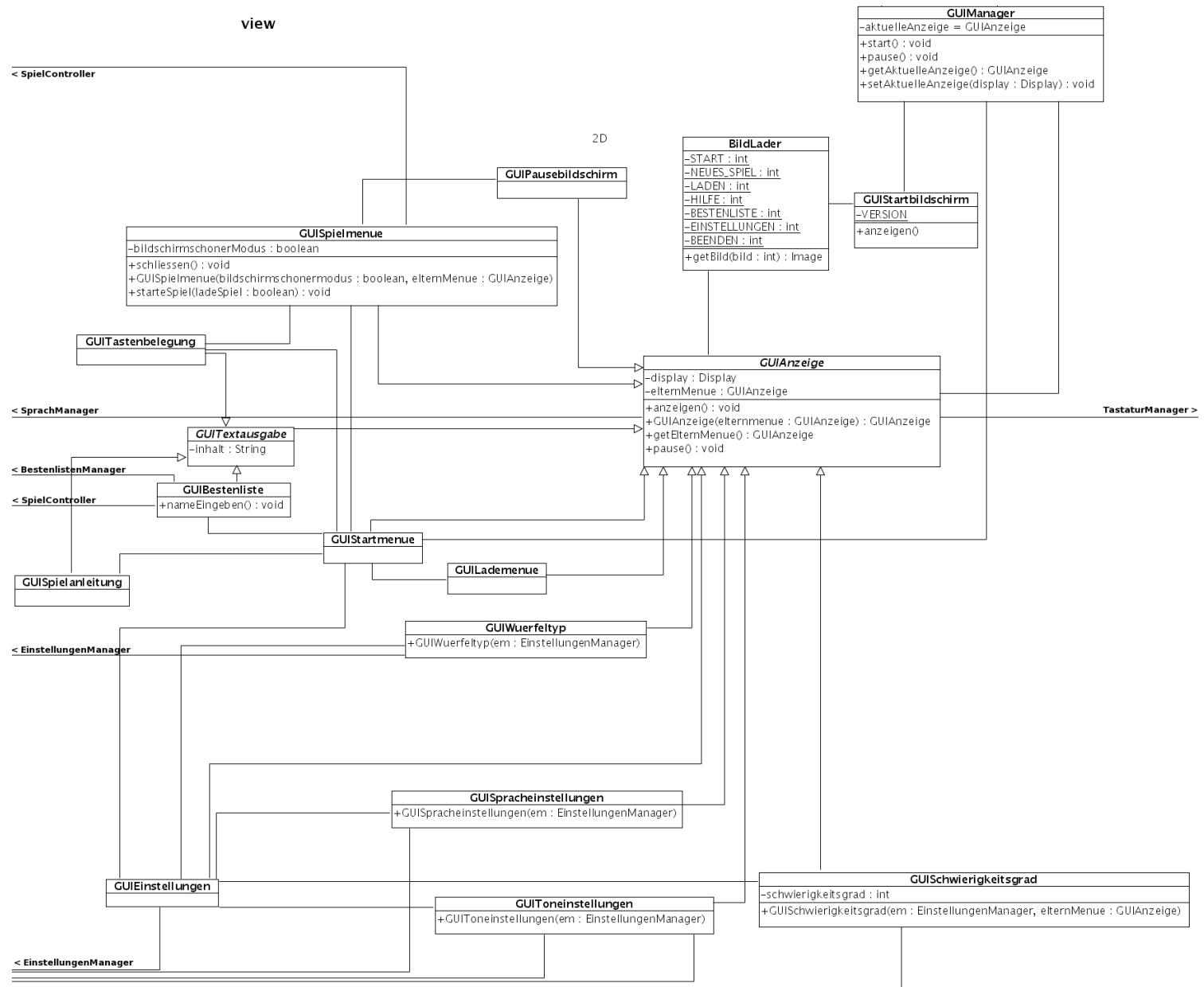


Abbildung 9 - 2D View

3 Klassenbeschreibung

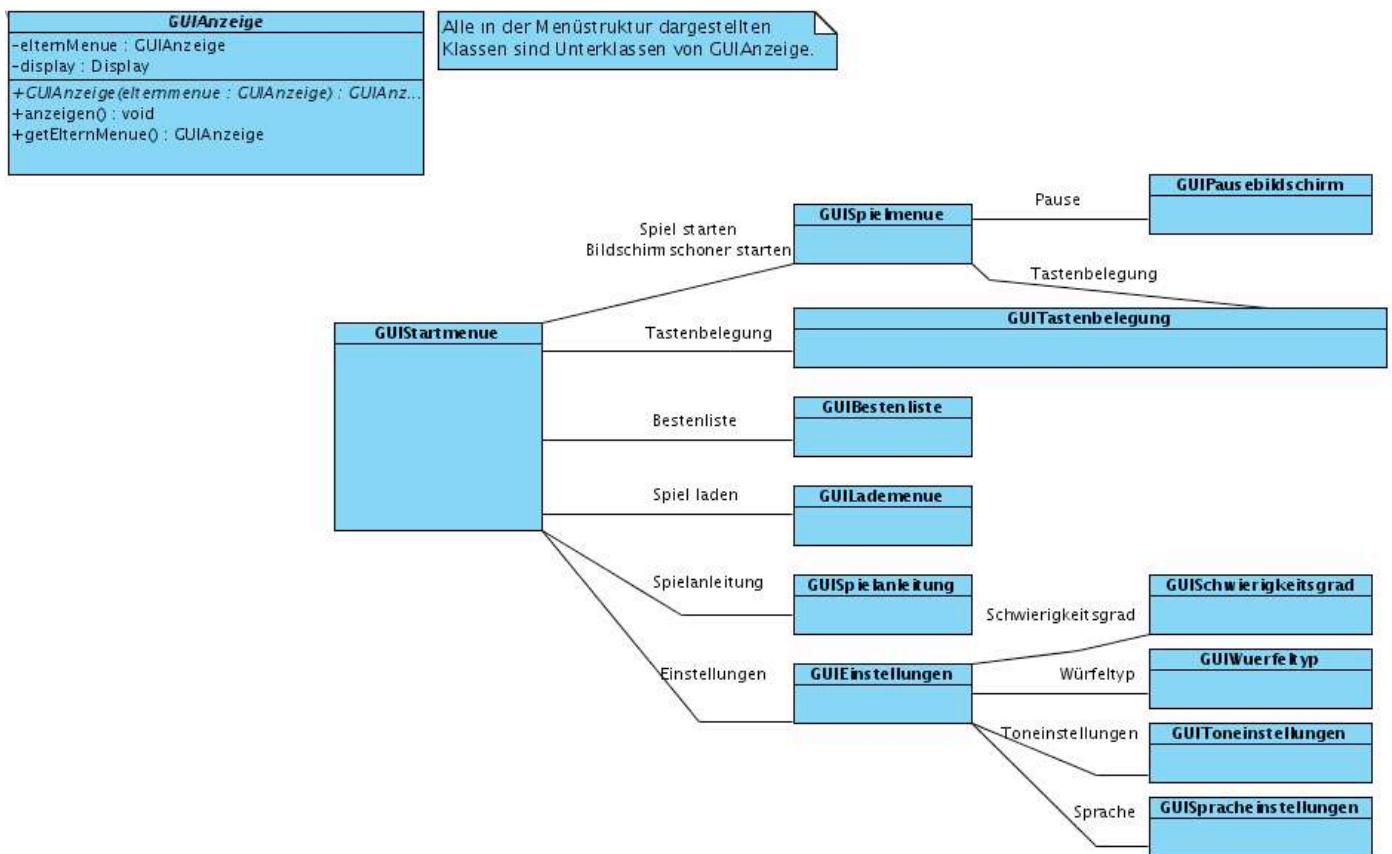


Abbildung 10 - Menüstruktur

► GUIStartbildschirm

GUIStartbildschirm
- VERSION : int
+ anzeigen()

Die Klasse GUIStartbildschirm stellt den Startbildschirm des Programmes auf dem Display dar.

Attribute

- *private static final int VERSION* Versionsnummer von YaRC.

Methoden

- *public anzeigen()* Zeigt den Startbildschirm an.

► GUIManager

GUIManager
+ aktuellesMenu : GUIAenzeige
+ start() : void
+ getAktuelleAnzeige() : GUIAenzeige
+ setAktuelleAnzeige(Display display) : void
+ pause() : void

Die Klasse GUIManager wird von der Klasse YaRCMIDlet aufgerufen und ruft ihrerseits die Klassen GUIStartbildschirm und GUIStartmenue auf, womit sie die Initialklasse für die grafische zweidimensionale Benutzeroberfläche ist.

Attribute

- *private GUIAenzeige aktuellesMenu* Das im Augenblick dargestellte Menü bzw. aktive GUI-Element.

Methoden

- *public void start()* Starten des GUIMangers.
- *public GUIAenzeige getAktuelleAnzeige()* Liefert das gerade aktive GUI-Element zurück.

- *public void setAktuelleAnzeige(Display display)* Sichert das gerade aktive GUI-Element.
- *public void pause()* Wird von YaRCMIDlet gerufen, wenn ein Anruf eingeht.

► GUIAnzeige

GUIAnzeige	
-	elternMenue : GUIAnzeige
-	display : Display
+	GUIAnzeige(GUIAnzeige elternmenue)
+	anzeigen() : void
+	getElternMenue() : GUIAnzeige
+	pause() : void

Die abstrakte Klasse GUIAnzeige fasst die Eigenschaften aller anzuzeigenden GUI-Elemente zusammen. über die Klasse BildLader werden Piktogramme für ggf. vorhandene Menüeinträge geladen. Die Klasse SprachManager versorgt die GUI-Elemente mit den der gewählten Sprache entsprechenden Textelementen. Via TastaturManager werden die Eingaben des Benutzers ausgelesen.

Attribute

- *private GUIAnzeige elternMenue* Referenz zum Elternmenü.
- *private Display display* Das Display, auf dem das jeweilige GUI-Element angezeigt wird.

Methoden

- *public GUIAnzeige(GUIAnzeige elternmenue)* Konstruktor mit übergebenen Elternmenü.
- *public void anzeigen()* Zeigt das Objekt auf dem Display an.
- *public GUIAnzeige getElternMenue()* Liefert das GUI-Element zurück, von dem das aktuelle aufgerufen wurde.
- *public void pause()* Sollte von jedem GUI-Element implementiert werden, um das Verhalten bei eingehendem Anruf zu steuern.

► GUIStartmenue

GUIStartmenue	

Die Klasse GUIStartmenue stellt das Startmenü auf dem Display dar und behandelt (über die von GUIAnzeige geerbte Assoziation zur Klasse TastaturManager) die Auswahl des Benutzers, indem sie das gewählte GUI-Element zur Anzeige bringt.

Attribute

- *geerbte Attribute* geerbte Attribute von GUIAnzeige.

Methoden

- *geerbte Methoden* geerbte Methoden von GUIAnzeige.

► GUISpielmenue

GUISpielmenue	
-	bildschirmschonerModus : boolean
+	GUISpielmenue(GUIAnzeige elternMenue, boolean bildschirmschonermodus)
+	starteSpiel(boolean ladeSpiel)

Die Klasse GUISpielmenue stellt den Eintrittspunkt in den Spielmodus dar. Sie steuert die Anzeige des Spielmenüs und leitet Benutzereingaben an den SpielController weiter.

Attribute

- *geerbte Attribute* geerbte Attribute von GUIAnzeige.

- *private boolean bildschirmschonerModus* Falls dieser Wert true ist befindet sich das Handyspiel im Bildschirmschonermodus.

Methoden

- *geerbte Methoden* geerbte Methoden von GUIAnzeige.
- *public GUISpielforme(GUIAnzeige elternMenue, boolean bildschirmschonermodus)* Erzeugt ein neues GUISpielforme Objekt. Falls der (zusätzlich zum Elternmenü) übergebene boolean-Wert true ist wird das Spiel im Bildschirmschonermodus gestartet.
- *public void starteSpiel(boolean ladeSpiel)* Falls der übergebene boolean-Wert true ist, wird das gespeicherte Spiel geladen. Falls kein Spiel gespeichert wurde, oder der boolean-Wert false ist wird ein neues und zufälliges Spiel erzeugt.

► GUILademenue

GUILademenue

Zeigt dem Benutzer an, ob ein gespeichertes Spiel vorhanden ist und ermöglicht ihm, dieses zu laden.

Attribute

- *geerbte Attribute* geerbte Attribute von GUIAnzeige.

Methoden

- *geerbte Methoden* geerbte Methoden von GUIAnzeige.

► GUITextausgabe

GUITextausgabe
- inhalt : String

Die abstrakte Klasse GUITextausgabe fasst als spezielle Klasse von GUIAnzeige die Eigenschaften der GUI-Elemente zusammen, die lediglich einfachen Text anzeigen.

Attribute

- *private String inhalt* Der formatierte Text.

Methoden

- *geerbte Methoden* geerbte Methoden von GUIAnzeige.

► GUIBestenliste

GUIBestenliste
- bestenliste : String
+ namenEingeben()

Die Klasse GUIBestenliste zeigt die aktuelle Bestenliste auf dem Display an. Die hierfür benötigten Daten holt sie sich vom BestenlisteManager.

Attribute

- *geerbte Attribute* geerbte Attribute von GUITextausgabe.

Methoden

- *geerbte Methoden* geerbte Methoden von GUITextausgabe.

- *public void namenEingeben()* Es erscheint am Bildschirm ein Fenster, in dem steht: "Herzlichen Glückwunsch! Sie haben den X.Platz erreicht und kommen somit in die Bestenliste. Geben Sie bitte Ihren Namen ein: _____." Dieser Name wird dann im BestenlisteManager gespeichert.

► GUI Spielanleitung

GUI Spielanleitung
- anleitung : String

Die Klasse GUI Spielanleitung zeigt die Spielanleitung an.

Attribute

- *geerbte Attribute* geerbte Attribute von GUITextausgabe.

Methoden

- *geerbte Methoden* geerbte Methoden von GUITextausgabe.

► GUITastenbelegung

GUITastenbelegung

Die Klasse GUITastenbelegung zeigt die Tastenbelegung für den Spielmodus an.

Attribute

- *geerbte Attribute* geerbte Attribute von GUITextausgabe.

Methoden

- *geerbte Methoden* geerbte Methoden von GUITextausgabe.

► GUIEinstellungen

GUIEinstellungen
geerbte Attribute - einstellungenManager : EinstellungenManager geerbte Methoden

Die Klasse GUIEinstellungen repräsentiert das aus dem Startmenü zugängliche Untermenü für die möglichen Einstellungen, die der Benutzer vornehmen kann. Die Einstellungen werden über den EinstellungenManager abgeglichen. Um dies zu ermöglichen, wird dem von GUIEinstellungen aufgerufenen Einstellungsmenü eine Referenz auf den EinstellungenManager mitgegeben.

Attribute

- *geerbte Attribute* geerbte Attribute von GUIAnzeige.
- *private EinstellungenManager einstellungenManager* Referenz auf den EinstellungenManager.

Methoden

- *geerbte Methoden* geerbte Methoden von GUIAnzeige.

► GUISpracheinstellungen

GUISpracheinstellungen
+ GUISchwierigkeitsgrad(EinstellungenManager em)

Zeigt die aktuellen Spracheinstellungen an und ermöglicht dem Benutzer, diese zu ändern.

Attribute

Methoden

- *public GUISchwierigkeitsgrad(EinstellungenManager em)* Dem Konstruktor wird die Referenz auf den EinstellungenManager übergeben.

► GUISchwierigkeitsgrad

GUISchwierigkeitsgrad
- schwierigkeitsgrad : int
+ GUISchwierigkeitsgrad(GUIAnzeige elternMenue, EinstellungenManager em)

Zeigt den aktuellen Schwierigkeitsgrad an und ermöglicht dem Benutzer, diesen zu ändern.

Attribute

- *private int schwierigkeitsgrad* Der aktuelle Schwierigkeitsgrad.

Methoden

- *public GUISchwierigkeitsgrad(GUIAnzeige elternMenue, EinstellungenManager em)* Dem Konstruktor wird zusätzlich die Referenz auf den EinstellungenManager übergeben.

► GUIToneinstellungen

GUIToneinstellungen
-
+ GUISchwierigkeitsgrad(EinstellungenManager em)

Zeigt die aktuellen Toneinstellungen an und ermöglicht dem Benutzer, diese zu ändern.

Attribute

Methoden

- *public GUISchwierigkeitsgrad(EinstellungenManager em)* Dem Konstruktor wird die Referenz auf den EinstellungenManager übergeben.

► GUIWuerfeltyp

GUIWuerfeltyp
-
+ GUISchwierigkeitsgrad(EinstellungenManager em)

Zeigt den aktuellen Würfeltyp an und ermöglicht dem Benutzer, diesen zu ändern.

Attribute

Methoden

- *public GUISchwierigkeitsgrad(EinstellungenManager em)* Dem Konstruktor wird die Referenz auf den EinstellungenManager übergeben.

► GUIPausebildschirm

GUIPausebildschirm
-
+

Diese Klasse zeigt den Pausebildschirm an.

Attribute

- *geerbte Attribute* geerbte Attribute von GUIAnzeige.

Methoden

- *geerbte Methoden* geerbte Methoden von GUIAnzeige.

► BildLader

BildLader
+ START : int + NEUES_SPIEL : int + LADEN : int + HILFE : int + BESTENLISTE : int + EINSTELLUNGEN : int + BEENDEN : int + getBild(int bild) : Image

Die Klasse BildLader ist für das Laden der Piktogramme für die einzelnen Menüs zuständig.

Attribute

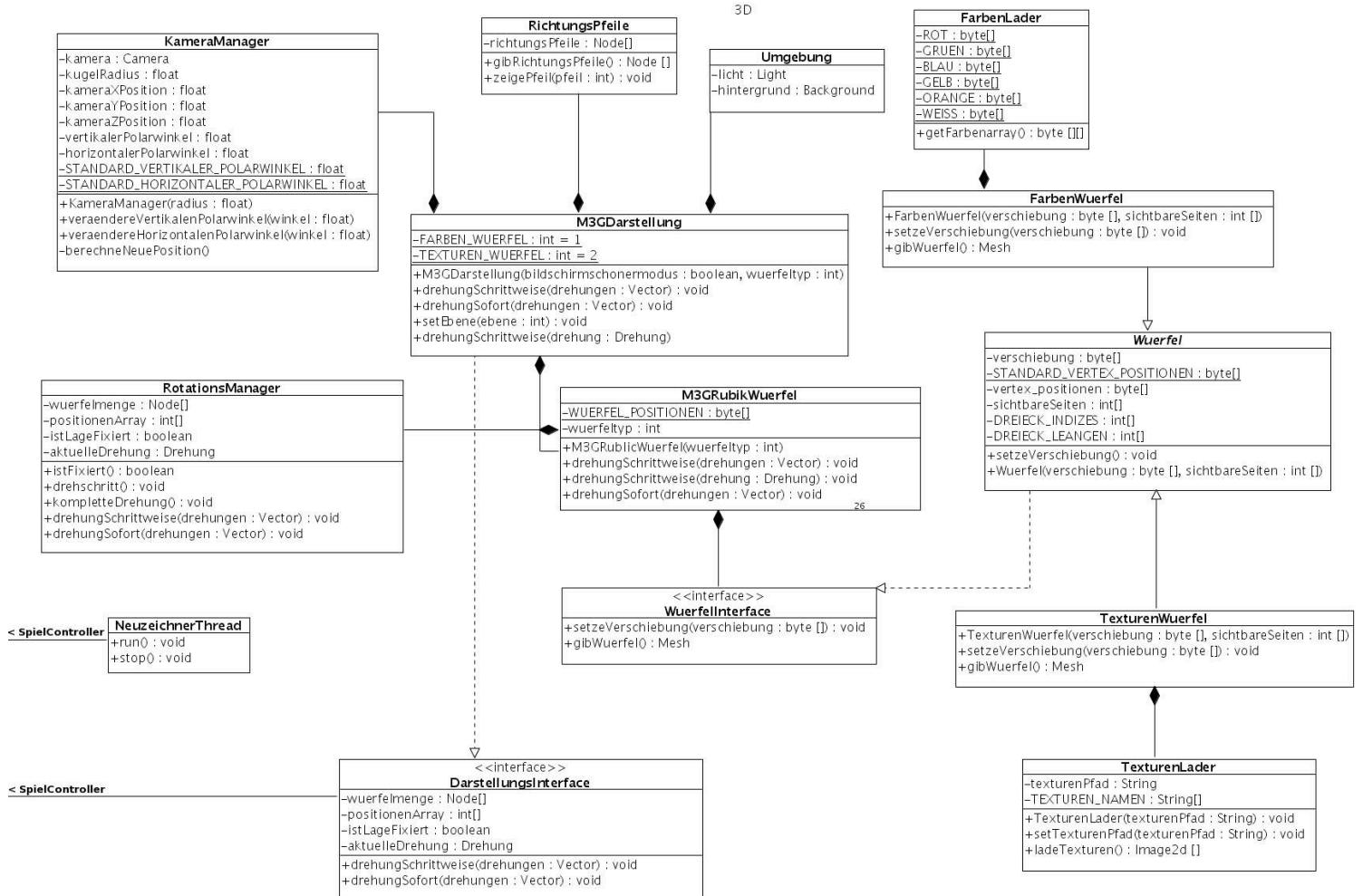
- *public static final int START* Konstante für das Startbild
- *public static final int NEUES_SPIEL* Konstante für das Piktogramm "Neues Spiel"
- *public static final int LADEN* Konstante für das Piktogramm "Laden"
- *public static final int HILFE* Konstante für das Piktogramm "Hilfe"
- *public static final int BESTENLISTE* Konstante für das Piktogramm "Bestenliste"
- *public static final int EINSTELLUNGEN* Konstante für das Piktogramm "Einstellungen"
- *public static final int BEENDEN* Konstante für das Piktogramm "Beenden"

Methoden

- *public Image getBild(int bild)* Liefert das entsprechende Bild zurück.

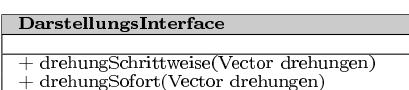
3 Klassenbeschreibung

3.4.2 3D View



Abbildungen 11 - 3D View

► DarstellungsInterface



Das DarstellungsInterface garantiert die Methoden, die dem Model bzw. Controller zur Verfügung stehen müssen, damit die Änderungen an dem Rubikwürfel dargestellt werden können.

Attribute Methoden

- *public void drehungSchrittweise(Vector drehungen)* Dreht die Würfelmenge animiert um die übergebene Drehung-Liste.
- *public void drehungSofort(Vector drehungen)* Dreht die Würfelmenge sofort um die übergebene Drehung-Liste.

► M3GDarstellung

M3GDarstellung
+ FARBEN_WUERFEL : int + TEXTUREN_WUERFEL : int - bildschirmschonerModus : boolean + M3GDarstellung(boolean bildschirmschonerModus, int wuerfeltyp) + setEbene(int ebene) + drehungSchrittweise(Vector drehungen) + drehungSchrittweise(Drehung drehung) + drehungSofort(Vector drehungen) + init() : void + paint() + keyPressed(int keyCode)

Die Klasse M3GDarstellung kombiniert alle Elemente der dreidimensionalen Welt und leitet die Drehungen an den M3GRubikWuerfel weiter.

Attribute

- *public static final int FARBEN_WUERFEL=1* Konstante für Farbenwürfel.
- *public static final int TEXTUREN_WUERFEL=2* Konstante für Texturenwürfel.
- *private boolean bildschirmschonerModus* Falls dieser Wert true ist befindet sich das Handyspiel im Bildschirmschonermodus.

Methoden

- *public M3GDarstellung(boolean bildschirmschonerModus, int wuerfeltyp)* Erzeugt ein neues GUI-Spielmenü-Objekt. Falls der übergebene boolean-Wert true ist wird das Spiel im Bildschirmschonermodus gestartet. Der int-Wert wuerfeltyp definiert den Würfeltyp (Farben- oder Texturenwürfel).
- *public void setEbene(int ebene)* Setzt eine gewählte Achse fest. Der Pfeil der Achse wird dann anschließend hervorgehoben. Diese Methode wird vom SpielController aufgerufen.
- *public void drehungSchrittweise(Vector drehungen)* Dreht die Würfelmenge animiert um die übergebene Drehung-Liste.
- *public void drehungSchrittweise(Drehung drehung)* Dreht die Würfelmenge animiert um die übergebene Drehung. (Overloading)
- *public void drehungSofort(Vector drehungen)* Dreht die Würfelmenge sofort um die übergebene Drehung-Liste.
- *protected void init()* Initialisiert die 3D-Anzeige.
- *protected void paint(Graphics graphics)* Zeichnet die komplette 3D-Welt.
- *protected void keyPressed(int keyCode)* Die Methode keyPressed leitet die Tastatureingaben zum SpielController weiter.

► M3GRubikWuerfel

M3GRubikWuerfel
- wuerfeltyp : int - WUERFEL_POSITIONEN : byte[] + M3GRubikWuerfel(int wuerfeltyp) + drehungSchrittweise(Vector drehungen) + drehungSchrittweise(Drehung drehung) + drehungSofort(Vector drehungen)

Die Klasse M3GRubikWuerfel implementiert einen dreidimensionalen Rubikwürfel bestehend aus einzelnen Würfeln. Dabei koordiniert diese Klasse die Positionierung und deligiert die Rotation der Teilwürfel.

Attribute

- *private int wuerfeltyp* Speichert den Würfeltyp.
- *private static final byte[] WUERFEL_POSITIONEN* Positionen der einzelnen Teilwürfel.

Methoden

- *public M3GRublicWuerfel(int wuerfeltyp)* Erzeugt ein neues M3GRubikWuerfel-Objekt mit übergebenem Würfeltyp.
- *public void drehungSchrittweise(Vector drehungen)* Die Methode drehungSchrittweise leitet die Drehung an den RotationsManager weiter.
- *public void drehungSchrittweise(Drehung drehung)* Die Methode drehungSchrittweise leitet die Drehung an den RotationsManager weiter. (Overloading)
- *public void drehungSofort(Vector drehungen)* Die Methode drehungSofort leitet die Drehungen an den RotationsManager weiter.

► Richtungspfeile

Richtungspfeile
- richtungsPfeile : Node[]
+ gibRichtungsPfeile : Node[]
+ zeigePfeil(int pfeil)

Die Klasse Richtungspfeile erzeugt für die M3GDarstellung neun Richtungspfeile mit Beschriftung. Diese dienen dazu dem Benutzer zu zeigen, mit welchen Tasteneingaben er welche Ebene in welche Richtung drehen kann.

Attribute

- *private Node[] richtungsPfeile* Die einzelnen Richtungspfeile.

Methoden

- *public Node[] gibRichtungsPfeile* Gibt die Richtungspfeile zurück.
- *public void zeigePfeil(int pfeil)* Hebt einen gewählten Pfeil hervor. Wird von M3GDarstellung aufgerufen.

► KameraManager

KameraManager
- kamera : Camera
- kugelRadius : float
- kameraXPosition : float
- kameraYPosition : float
- kameraZPosition : float
- vertikalerPolarwinkel : float
- horizontalerPolarwinkel : float
- STANDARD_VERTIKALER_POLARWINKEL : float
- STANDARD_HORIZONTALER_POLARWINKEL : float
+ KameraManager(float radius)
+ veraendereVertikalenPolarwinkel(float winkel)
+ veraendereHorizontalenPolarwinkel(float winkel)
+ berechneNeuePosition()

Der KameraManager realisiert sowohl die Kamera als auch ihre Funktionen. Das Bild der Kamera wird im Handydisplay dargestellt. Der Standort der Kamera lässt sich auf der Oberfläche einer virtuellen Kugel verschieben. Das Ziel der Kamera ist immer auf das Zentrum des Rubikwürfels gerichtet.

Attribute

- *private Camera kamera* M3G-Kamera Objekt.
- *private float kugelRadius* Radius der virtuellen Kugel (= Abstand der Kamera vom Würfelmittelpunkt).
- *private float kameraXPosition* Absolute X-Position der Kamera.
- *private float kameraYPosition* Absolute Y-Position der Kamera.
- *private float kameraZPosition* Absolute Z-Position der Kamera.
- *private float vertikalerPolarwinkel* Vertikaler Polarwinkel zum Kamerastandort.
- *private float horizontalerPolarwinkel* Horizontaler Polarwinkel zum Kamerastandort.

- *private static final float STANDARD_VERTIKALER_POLARWINKEL* Initialer vertikaler Polarwinkel zum Kamerastandort.
- *private static final float STANDARD_HORIZONTALER_POLARWINKEL* Initialer horizontaler Polarwinkel zum Kamerastandort.

Methoden

- *public KameraManager(float radius)* Der Konstruktor erzeugt ein neues KameraManager-Objekt.
- *public void veraendereVertikalenPolarwinkel(float winkel)* Verändert den vertikalen Polarwinkel um den übergebenen float-Wert.
- *public void veraendereHorizontalenPolarwinkel(float winkel)* Verändert den horizontalen Polarwinkel um den übergebenen float-Wert.
- *private void berechneNeuePosition()* Berechnet anhand der Polarwinkel die (X,Y,Z)Position der Kamera.

► Umgebung

Umgebung
- licht : Light - hintergrund : Background

Die Klasse Umgebung implementiert zusätzliche Objekte für die Ausgabe. Solche Objekte sind z.B. das Licht oder der Hintergrund.

Attribute

- *private Light licht* Globales Licht für die 3D-Welt.
- *private Background hintergrund* Hintergrund der 3D-Welt.

Methoden

► RotationsManager

RotationsManager
- wuerfelmenge : Node[] - positionenArray : int[] - istLageFixiert : boolean - aktuelleDrehung : Drehung + drehungSchrittweise(Vector drehungen) + drehungSofort(Vector drehungen) + istFixiert() : boolean - drehschritt() - kompletteDrehung()

Die Klasse RotationsManager implementiert die Drehung der einzelnen Teilwürfel der View. Es wird dabei nach jeder abgeschlossenen Drehung die neue Position der Teilwürfel aktualisiert. Die Klasse bietet außerdem eine Funktion, eine übergebene Liste an Drehungen für die Würfelerzeugung abarbeitet.

Attribute

- *private Node[] wuerfelmenge* Referenz auf die komplette Menge der Würfel, aus denen der Rubik-Würfel aufgebaut ist.
- *private int[] positionenArray* Position jedes einzelnen Würfels in der Würfelmenge.
- *private boolean istLageFixiert* Speichert, ob eine Drehung abgeschlossen ist, oder nicht.
- *private Drehung aktuelleDrehung* Speichert die aktuelle Drehung.

Methoden

- *public void drehungSchrittweise(Vector drehungen)* Dreht die Würfelmenge animiert um die übergebene Drehung-Liste
- *public void drehungSofort(Vector drehungen)* Dreht die Würfelmenge sofort um die übergebene Drehung-Liste
- *public boolean istFixiert()* Gibt zurück, ob die vorhergegangene Drehung abgeschlossen ist, oder nicht.

3 Klassenbeschreibung

- *private void drehschritt()* Führt einen einzelnen Schritt der Teildrehung aus.
- *private void kompletteDrehung()* Führt eine komplette Drehung auf einmal aus.

► WuerfelInterface

WuerfelInterface
+ setzeVerschiebung(byte[] verschiebung)
+ gibWuerfel() : Mesh

Das WuerfelInterface definiert die Methoden, die Klassen implementieren müssen, die einen dreidimensionalen Würfel implementieren.

Attribute

Methoden

- *public void setzeVerschiebung(byte[] verschiebung)* Der Würfel soll um das übergebene 'byte'-Array im Raum verschoben werden.
- *public Mesh gibWuerfel()* Gibt den aktuellen Würfel als Mesh zurück.

► Wuerfel

Wuerfel
- verschiebung : byte[]
- STANDARD_VERTEX_POSITIONEN : byte[]
- vertex_positionen : byte[]
- sichtbareSeiten : int[]
- DREIECK_INDIZES : int[]
- DREIECK_LEANGEN : int[]
+ void setzeVerschiebung
+ Wuerfen(byte[] verschiebung,int[] sichtbareSeiten)

Die abstrakte Klasse Wuerfel definiert einen allgemeinen Würfel, der aus dreidimensionalen Gitterlinien aufgebaut ist und jeder Seite eine bestimmte Farbe/Textur zuordnen kann.

Attribute

- *private byte[] verschiebung* Verschiebung des Würfels im dreidimensionalen Raum.
- *private static final byte[] STANDARD_VERTEX_POSITIONEN* Standardpositionen des Eckpunkte des Würfels.
- *private byte[] vertex_positionen* Positionen der einzelnen Vertex-Flächen.
- *private int[] sichtbareSeiten* Seiten des Würfels, die nicht von einem anderen Würfel verdeckt werden.
- *private int[] DREIECK_INDIZES*
- *private int[] DREIECK_LEANGEN*

Methoden

- *public void setzeVerschiebung*
- *public Wuerfen(byte[] verschiebung, int[] sichtbareSeiten)* Der Standardkonstruktor der Klasse Wuerfel, der einen neuen Würfel mit übergebener Verschiebung erzeugt.

► FarbenWuerfel

FarbenWuerfel
+ FarbenWuerfel(byte[] verschiebung,int[] sichtbareSeiten)
+ setzeVerschiebung(byte[] verschiebung)
+ gibWuerfel() : Mesh

Die Klasse FarbenWuerfel implementiert einen dreidimensionalen Würfel mit RGB-Farben auf seiner Oberfläche.

Attribute

- Nur geerbte Attribute von *Wuerfel*

Methoden

- *public FarbenWuerfel(byte[] verschiebung, int[] sichtbareSeiten)* überschreibt den Standard-Konstruktor der Wuerfel-Klasse und erzeugt einen neuen Farbenwürfel.
- *public void setzeVerschiebung(byte[] verschiebung)* Der Würfel soll um das übergebene 'byte'-Array im Raum verschoben werden.
- *public Mesh gibWuerfel()* Gibt den aktuellen Würfel als Mesh zurück.

► TexturenWuerfel

TexturenWuerfel
+ TexturenWuerfel(byte[] verschiebung, int[] sichtbareSeiten)
+ setzeVerschiebung(byte[] verschiebung)
+ gibWuerfel() : Mesh

Die Klasse TexturenWuerfel implementiert einen dreidimensionalen Würfel mit Texturen auf seiner Oberfläche.

Attribute

- Nur geerbte Attribute von *Wuerfel*

Methoden

- *public TexturenWuerfel(byte[] verschiebung, int[] sichtbareSeiten)* überschreibt den Standard-Konstruktor der Wuerfel-Klasse und erzeugt einen neuen Texturenwürfel.
- *public void setzeVerschiebung(byte[] verschiebung)* Der Würfel soll um das übergebene 'byte'-Array im Raum verschoben werden.
- *public Mesh gibWuerfel()* Gibt den aktuellen Würfel als Mesh zurück.

► FarbenLader

FarbenLader
- ROT : byte[]
- GRUEN : byte[]
- BLAU : byte[]
- GELB Farbe Gelb : byte[]
- ORANGE : byte[]
- WEISS byte[]
+ getFarbenarray : byte[][]

Die Klasse FarbenLader definiert die einzelnen Farben, die ein dreidimensionaler Würfel auf jeder Seite hat. Da beim Farbenwürfel auf jeder Seitenfläche für jede der vier Ecken eine Farbe angegeben werden muss, benötigt der Würfel $6 * 4 = 24$ Farbangaben.

Attribute

- *private static final byte[] ROT* Farbe Rot
- *private static final byte[] GRUEN* Farbe Grün
- *private static final byte[] BLAU* Farbe Blau
- *private static final byte[] GELB* Farbe Gelb
- *private static final byte[] ORANGE* Farbe Orange
- *private static final byte[] WEISS* Farbe Weiß

Methoden

- *public byte[][] getFarbenarray* Die Methode getFarbenarray gibt ein zweidimensionales Array mit allen Farben des Würfels auf der jeweiligen Seite zurück.

► TexturenLader

TexturenLader

- texturenPfad : String
- TEXTUREN_NAMEN : String[]
+ TexturenLader(String texturenPfad)
+ setTexturenPfad(String pfad)
+ ladeTexturen : Image2d[]

Die Klasse TexturenLader lädt sechs Texturen für den texturierten Rublikwürfel.

Attribute

- *private String texturenPfad* Verzeichniss in dem sich die Texturen befinden.
- *private static final String[] TEXTUREN_NAMEN* Die festen Dateinamen der einzelnen Texturdateien für den Würfel.

Methoden

- *public TexturenLader(String texturenPfad)* Standard- Konstruktor mit übergebenen Parameter texturenPfad: Verzeichnis in dem sich die Texturen befinden.
- *public void setTexturenPfad(String texturenPfad)* Setzt den Wert des Attributes texturenPfad.
- *public Image2d[] ladeTexturen* Gibt ein Image2d- Array mit den jeweiligen Texturen für jede Seite zurück.

► NeuzeichnerThread

NeuzeichnerThread

+ run()
+ stop()

Die Klasse NeuzeichnerThread implementiert einen TimerTask für das fortlaufende Neuzeichnen der Canvs. Auf dieser Canvs wird der 3D-Würfel dargestellt.

Attribute

Methoden

- *public void run()* Startet das Neuzeichnen.
- *public void stop()* Stoppt das Neuzeichnen.

4 Abläufe

4.1 Spielstart

Dieses Sequenzdiagramm zeigt den Start eines neuen Spiels von der GUI-Spielmenü-Klasse aus. Dazu wird zuerst eine neue Instanz der Klasse SpielController erzeugt, die während des Spiel die zentrale Instanz darstellt. Der SpielController erzeugt sich seine Hilfsklassen (z.B. ZeitManager), holt sich einen neuen Würfel vom StartGenerator und übergibt diesen an das Model (RubikWuerfel) und die View (M3GDarstellung). Anschließend wird die 3D-Welt angezeigt und die Zeitmessung gestartet.

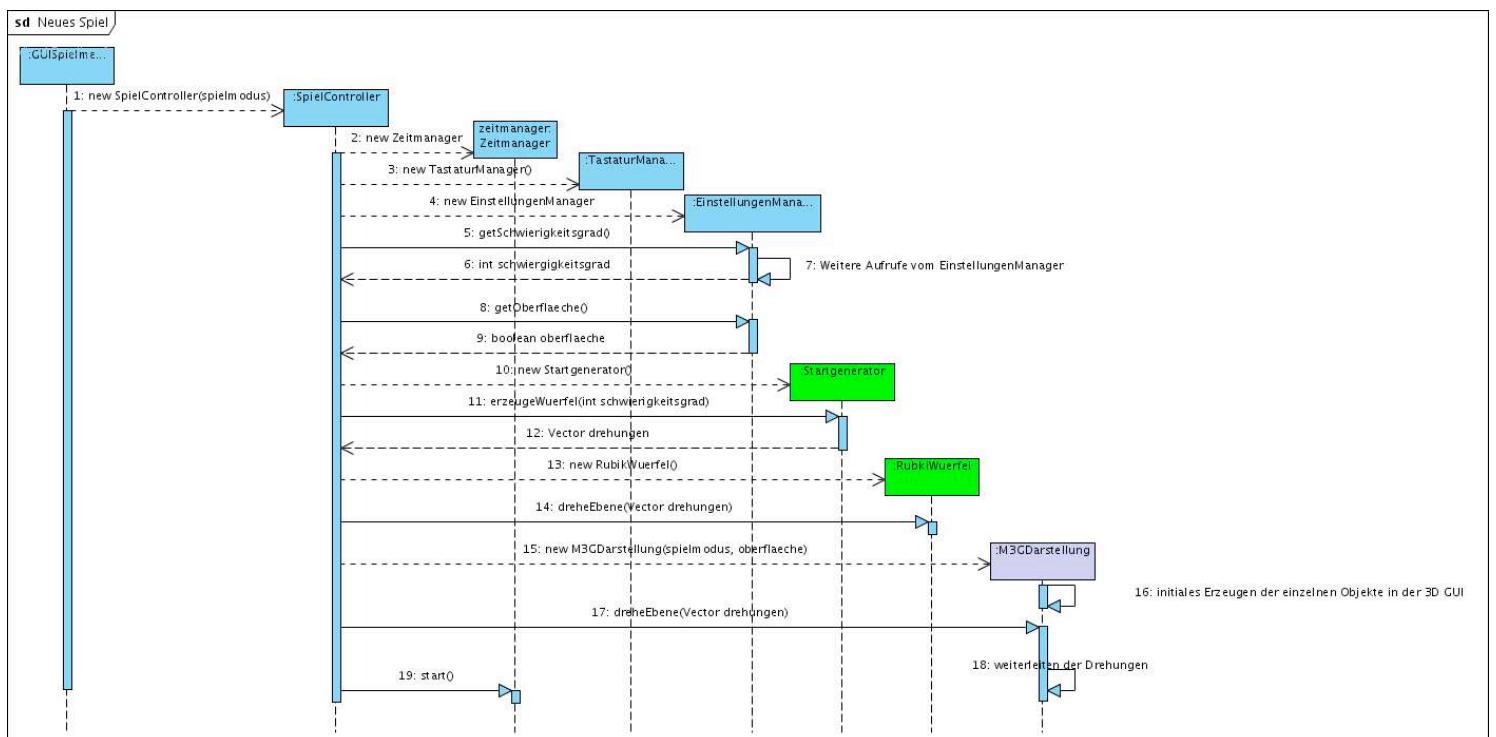


Abbildung 12 - Sequenz-Diagramm 1 : Spielstart

4 Abläufe

4.2 Programmstart und Änderung des Schwierigkeitsgrades

In diesem Sequenzdiagramm wird dargestellt, was geschieht, wenn der Benutzer das Programm startet und dann im Startmenü den Eintrag "Einstellungen" auswählt, um von dort aus den Schwierigkeitsgrad zu ändern. Im Anschluss daran geht er zurück ins Startmenü und wählt "Spiel starten".

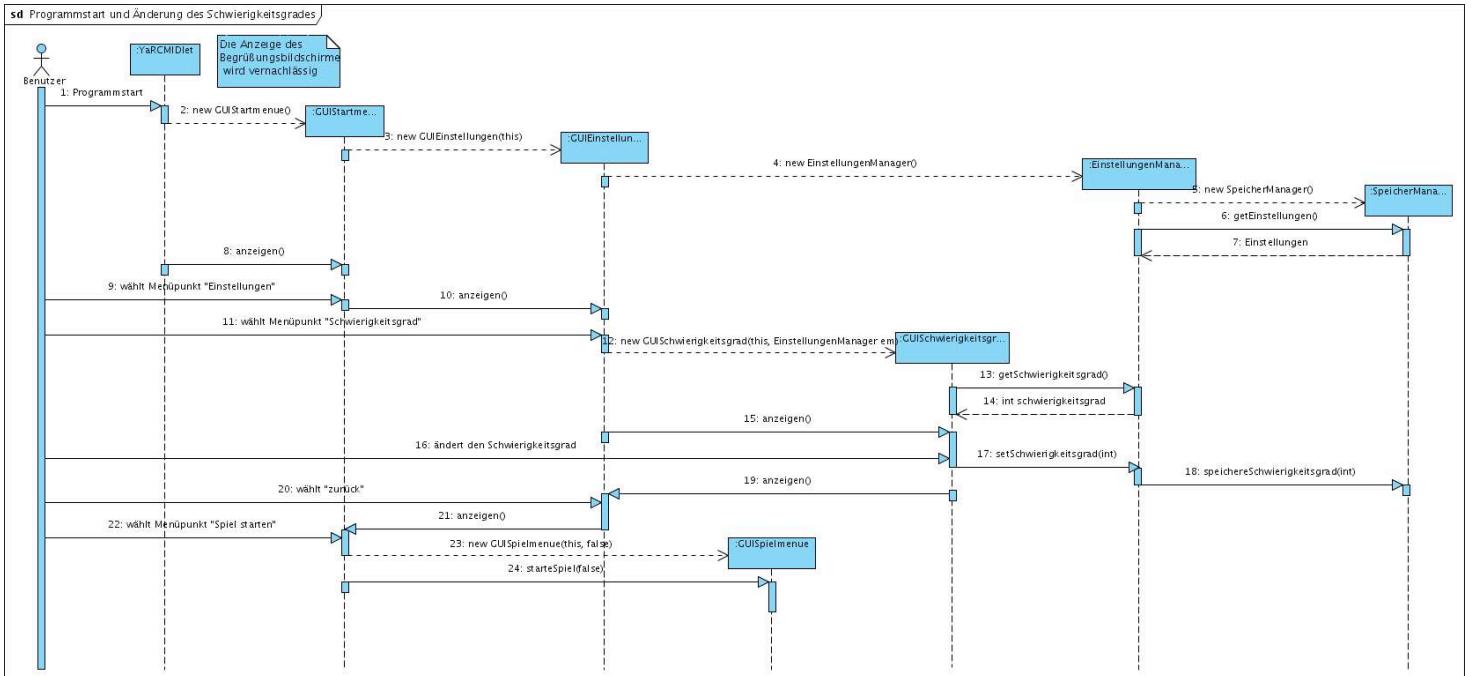


Abbildung 13 - Sequenz-Diagramm 2 : Programmstart und Änderung des Schwierigkeitsgrades

4.3 Drehung

In diesem Sequenzdiagramm wird beschrieben, wie eine durch einen Benutzer veranlasste Drehung abläuft. Nachdem der Benutzer die zu drehende Ebene ausgewählt hat, wird der zugehörige Pfeil in der Darstellung hervorgehoben. Sobald der Benutzer auch die Richtung gewählt hat, wird ein neues Objekt der Klasse Drehung erzeugt. Dieses übergibt anschließend der SpielController (Controller) an die Klassen RubikWuerfel und HistorieManager (Model), um die Drehung in der Datenstruktur auszuführen und die Drehung in die undo-Liste aufzunehmen. Anschließend wird vom SpielController die M3GDarstellung (View) aufgerufen, um die Drehung für den Benutzer sichtbar zu machen.

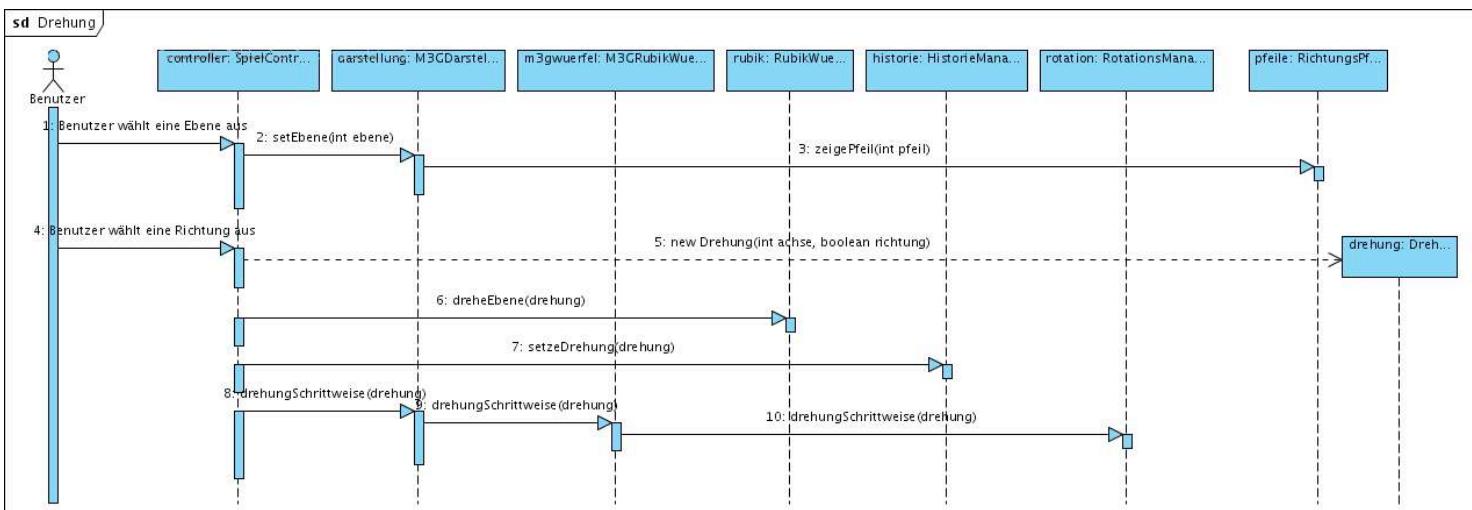


Abbildung 14 - Sequenz-Diagramm 3 : Drehung

4.4 Hilfefunktion

Hier wird der Ablauf beim Anwenden der Hilfe-Funktion dargestellt.

Nachdem der Benutzer das Spielmenü aufgerufen hat und die Hilfe-Funktion ausgewählt hat, wird von der Klasse GUISpielenue (View) die Methode getHilfe() von der Klasse SpielController (Controller) aufgerufen. Dieser fordert daraufhin den nächsten Spielzug von IntuitiverLoeser (Model), welcher eine Liste von Drehungen zurück gibt. Diese Drehungen werden dann analog wie im Sequenzdiagramm "Drehungen" ausgeführt.

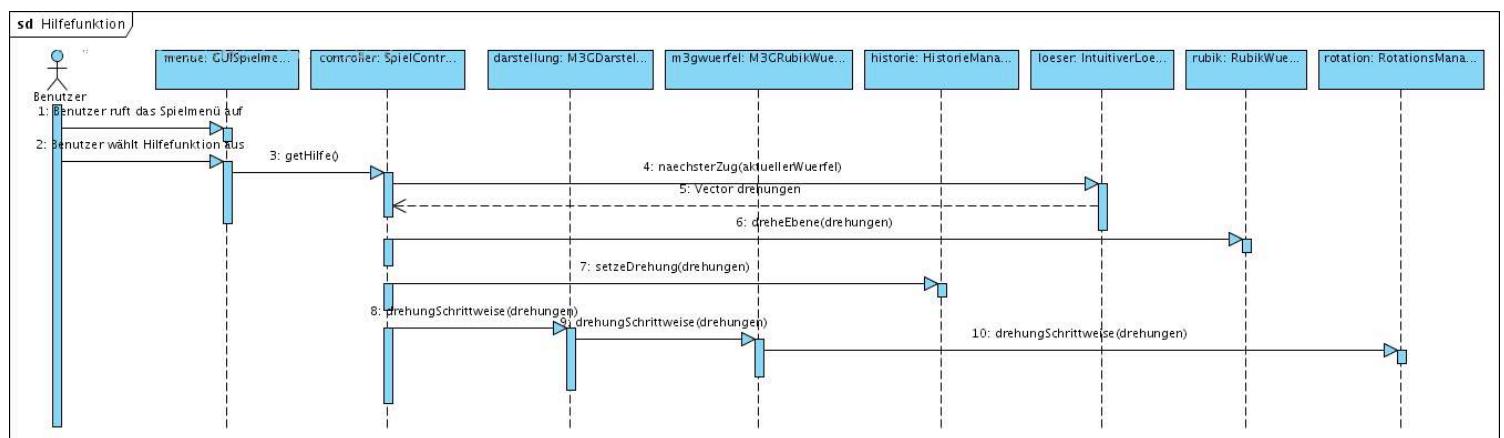


Abbildung 15 - Sequenz-Diagramm 4 : Hilfefunktion

4.5 Pause durch Anruf

Hier wird beschrieben, wie das Spiel durch einen Anruf unterbrochen wird.

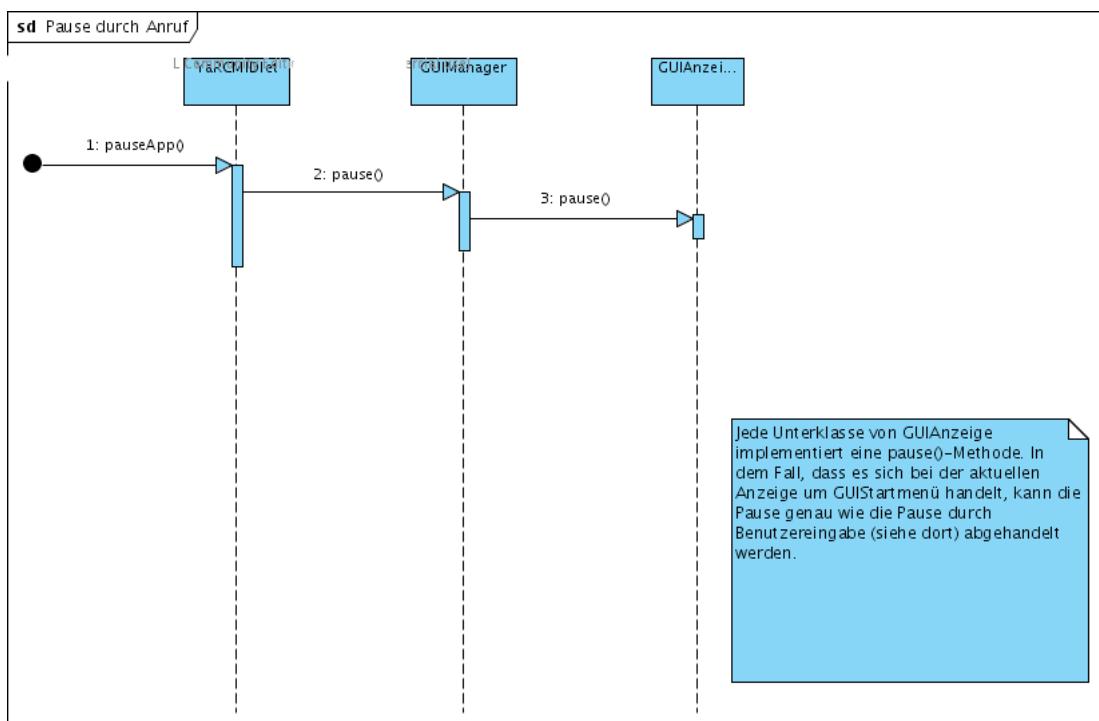


Abbildung 16 - Sequenz-Diagramm 5 : Pause durch Anruf

4.6 Pause durch Benutzer

Ausgangssituation für dieses Sequenzdiagramm ist der Spielmodus. Der Benutzer ruft das Spielmenü auf und wählt den Eintrag "Pause". Daraufhin wird die Spielzeit angehalten, so lange, bis der Benutzer die Pause beendet und das Spiel fortsetzt.

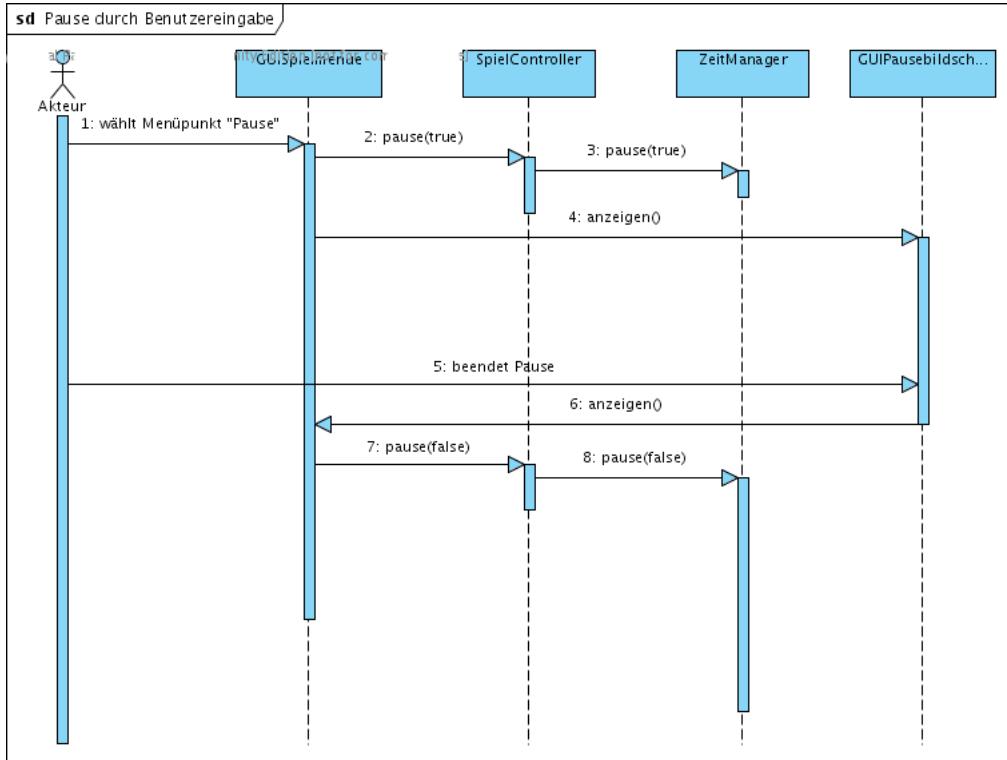


Abbildung 17 - Sequenz-Diagramm 6 : Pause durch Benutzer

4.7 Spiel gelöst

Im folgenden Sequenzdiagramm wird beschrieben, was sich ereignet, wenn der Rubikwürfel im Spielmodus gelöst wurde.

Zuerst veranlasst der SpielController die Zeit zu stoppen, als nächstes wird mit der benötigten Spielzeit und dem eingestellten Schwierigkeitsgrad die erreichten Punkte berechnet. Der BestenlisteManager vergleicht die erreichten Punkte mit denen, die bereits in der Bestenliste eingetragen sind, und teilt dann den SpielController mit, ob mit diesen Punkten der Highscore geknackt wurde und ob der Spieler einen Eintrag in die Bestenliste erhält.

In diesem Sequenzdiagramm wird angenommen, dass der Spieler einen Eintrag in die Bestenliste erhält, aber nicht den 1.Platz erzielt hat.

Nachdem dann die Standardmusik abgespielt wurde, wird der Benutzer aufgefordert seine Namen einzugeben. Die Bestenliste wird aktualisiert und das Startmenue angezeigt.

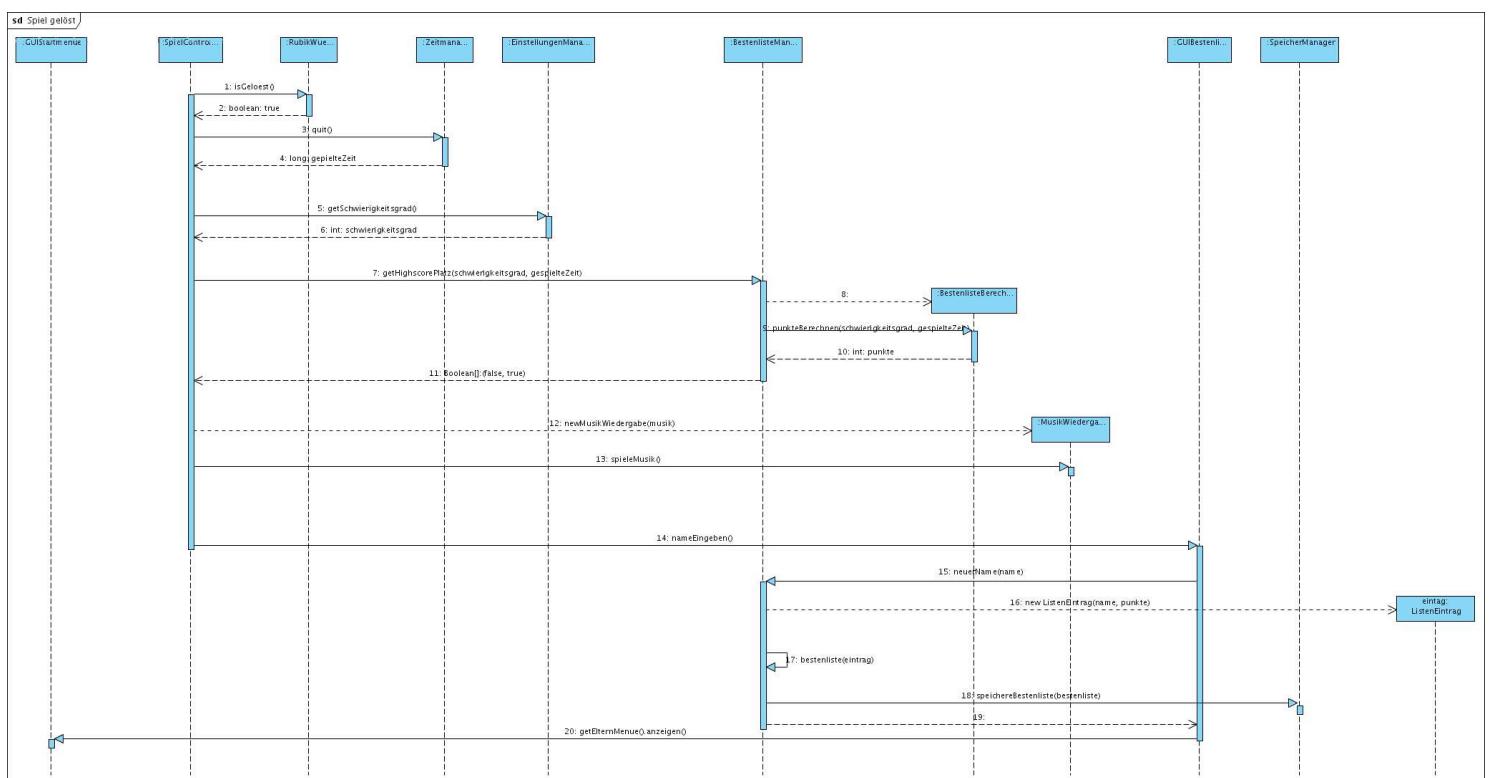


Abbildung 18 - Sequenz-Diagramm 7 : Spiel gelöst

4.8 Speichern und beenden

Dieses Sequenzdiagramm stellt den Programmablauf beim Speichern und beenden dar.

Der Benutzer wählt die Funktion Speichern und beenden im Spielmenü aus. Dazu muss der aktuelle Spielstand - bisher benötigte Zeit, Schwierigkeitsgrad, aktueller Würfel - gesichert werden. Danach wird das Startmenü wieder angezeigt.

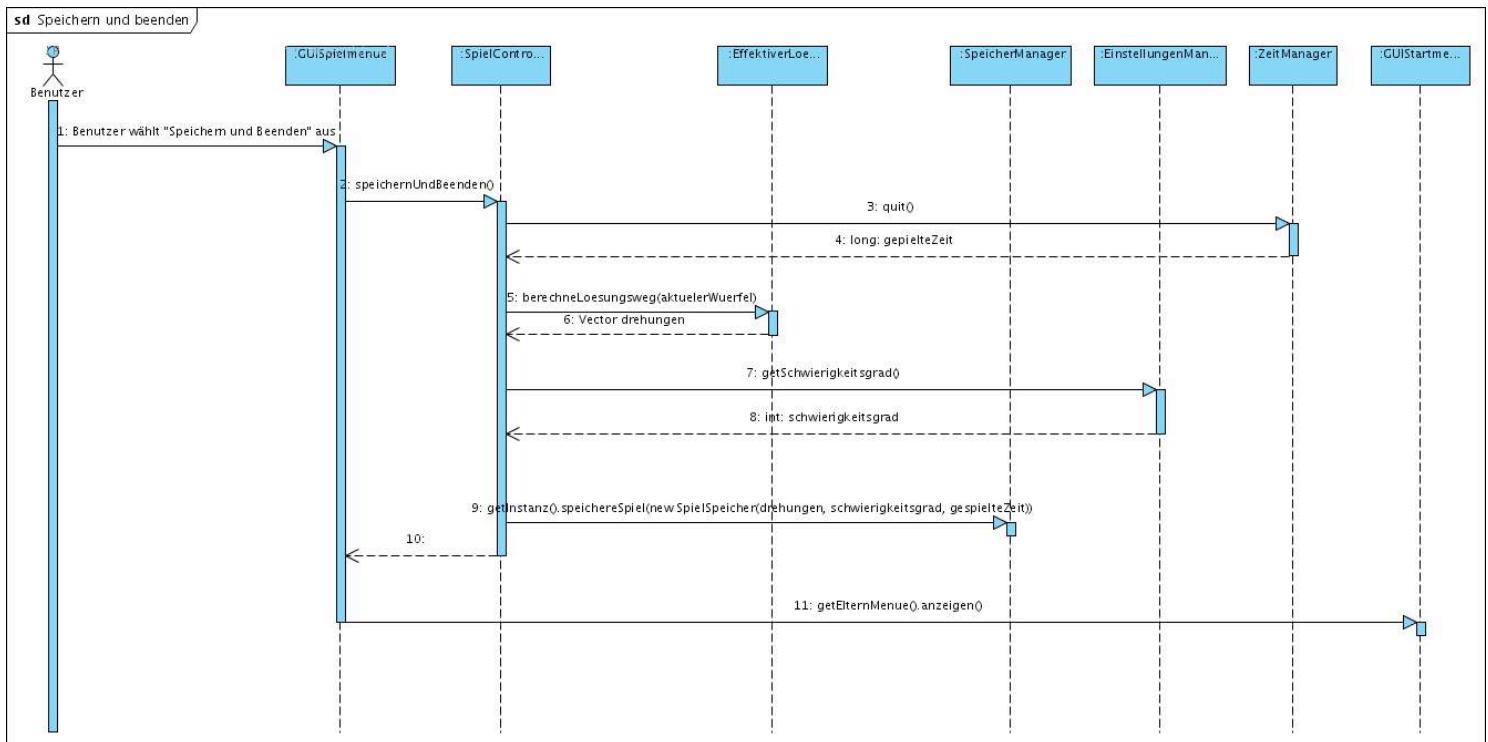


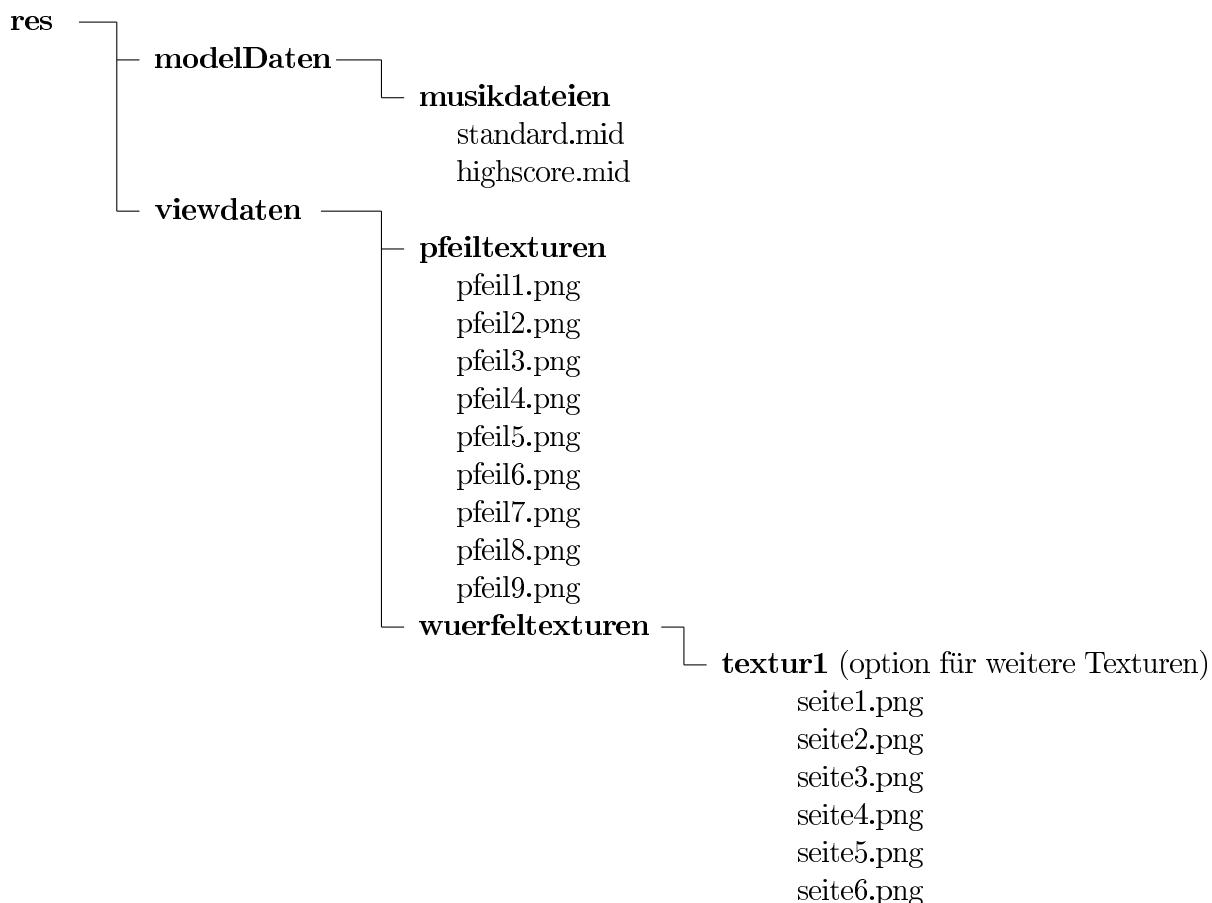
Abbildung 19 - Sequenz-Diagramm 8 : Speichern und beenden

5 Entwurfsdaten

In diesem Kapitel werden weiter wichtige Bestandteile des Entwurfs angegeben.

5.1 Ressourcen-Verzeichnis

In diesem Kapitel wird beschrieben, welche Daten in welchem Unterverzeichniss des **res**-Ordners gespeichert werden.



5.2 Dateneinträge

Da J2ME anders als J2SE keine Reflection und folglich keine Serialisierung unterstützt und kein Schreibzugriff auf das Filesystem des Handy's erlaubt ist, wird ein anderes Konzept gewählt, um veränderliche Daten zu sichern: der so genannte RecordStore.

Im RecordStore werden folgende Daten gesichert:

- Spielstand
- Einstellungen
- Sprachdateien
- Bestenliste

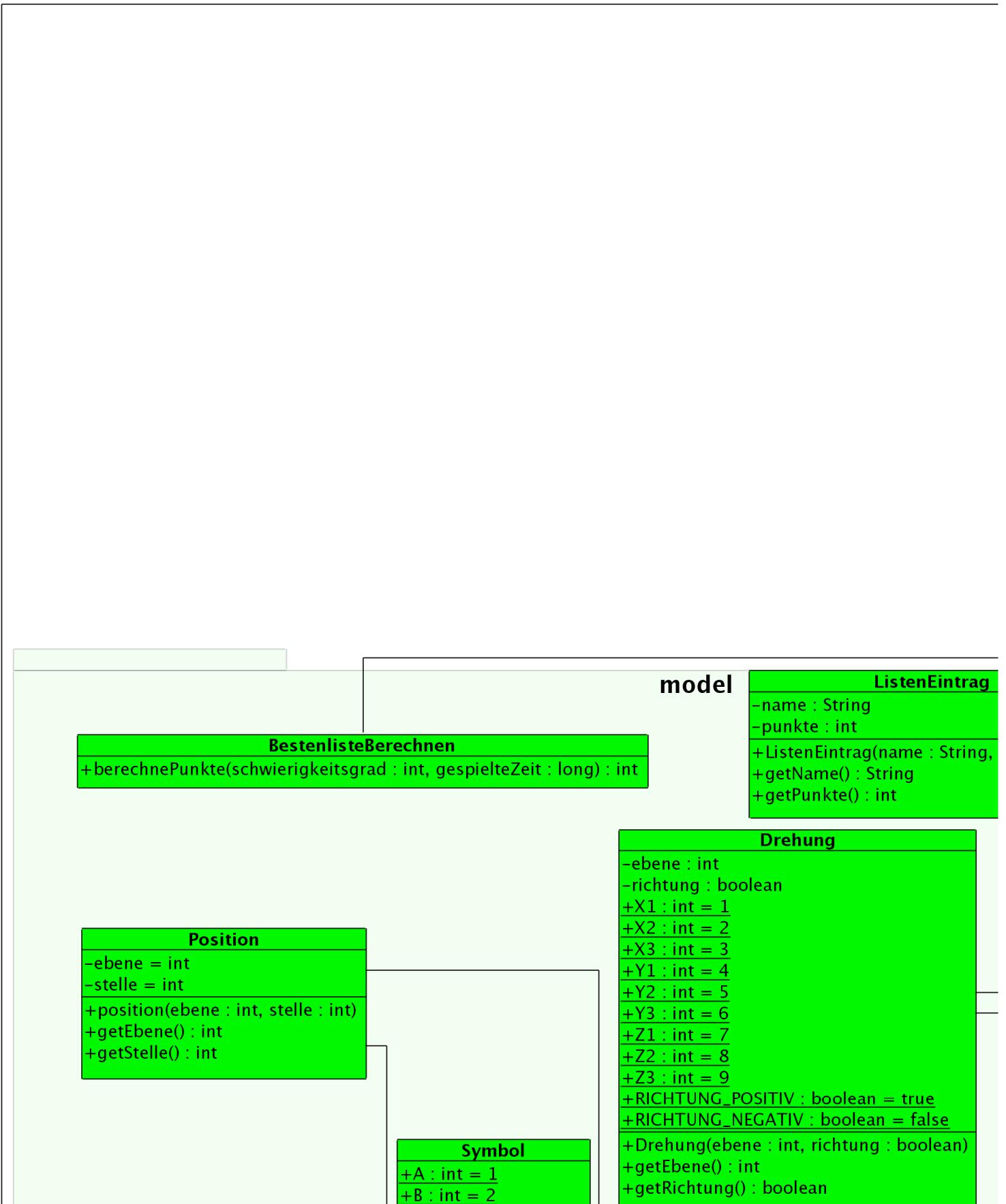
6 Klassenindex

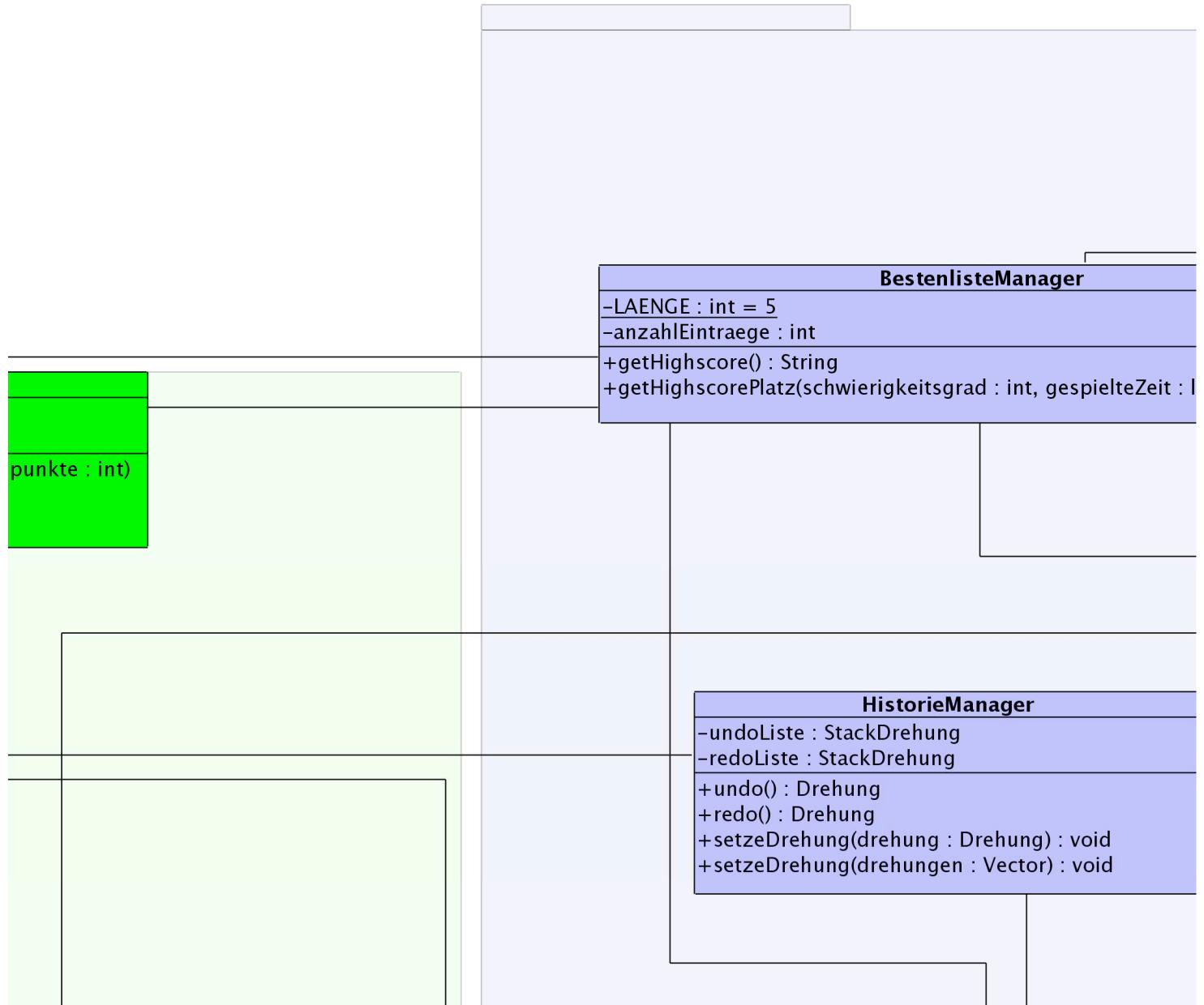
Alphabetisches Verzeichnis der Klassen.

Klasse	Position	Seite	Klasse	Position	Seite
BildLader	View:2D View	29	Wuerfel	View:3D View	34
DarstellungsInterface	View:3D View	30	WuerfelInterface	View:3D View	34
FarbenLader	View:3D View	35	BestenlisteBerechnen	Model	11
FarbenWuerfel	View:3D View	34	BestenlisteManager	Controller	21
GUIAnzeige	View:2D View	25	Drehung	Model	12
GUIBestenliste	View:2D View	26	EckWuerfel	Model	9
GUIEinstellungen	View:2D View	27	EffizienterLoeser	Model	10
GUILademenue	View:2D View	26	EinstellungenManager	Model	17
GUIManager	View:2D View	24	HistorieManager	Controller	20
GUIPausebildschirm	View:2D View	28	IntuitiverLoeser	Model	11
GUISchwierigkeitsgrad	View:2D View	28	KantenWuerfel	Model	9
GUISpielanleitung	View:2D View	27	ListenEintrag	Model	11
GUISpielmenue	View:2D View	25	LoeserInterface	Model	10
GUISpracheinstellungen	View:2D View	27	MittelWuerfel	Model	10
GUISstartbildschirm	View:2D View	24	MusikWiedergabe	Controller	22
GUISstartmenue	View:2D View	25	Position	Model	13
GUITastenbelegung	View:2D View	27	RubikWuerfel	Model	8
GUITextausgabe	View:2D View	26	SpeicherManager	Controller	22
GUIToneinstellungen	View:2D View	28	SpielController	Controller	16
GUIWuerfeltyp	View:2D View	28	SpielSpeicher	Controller	12
KameraManager	View:3D View	32	SprachManager	Controller	21
M3GDarstellung	View:3D View	31	Startgenerator	Model	11
M3GRubikWuerfel	View:3D View	31	Symbol	Model	14
NeuzeichnerThread	View:3D View	36	TastaturManager	Controller	18
Richtungspfeile	View:3D View	32	Teilwuerfel	Model	8
RotationsManager	View:3D View	33	XMLManager	Controller	20
TexturenLader	View:3D View	36	YaRCMIDlet	Controller	16
TexturenWuerfel	View:3D View	35	ZeitManager	Controller	19
Umgebung	View:3D View	33			

7 Anhang

Klassendiagramm auf 12 Seiten verteilt (ohne überlappung).





controller

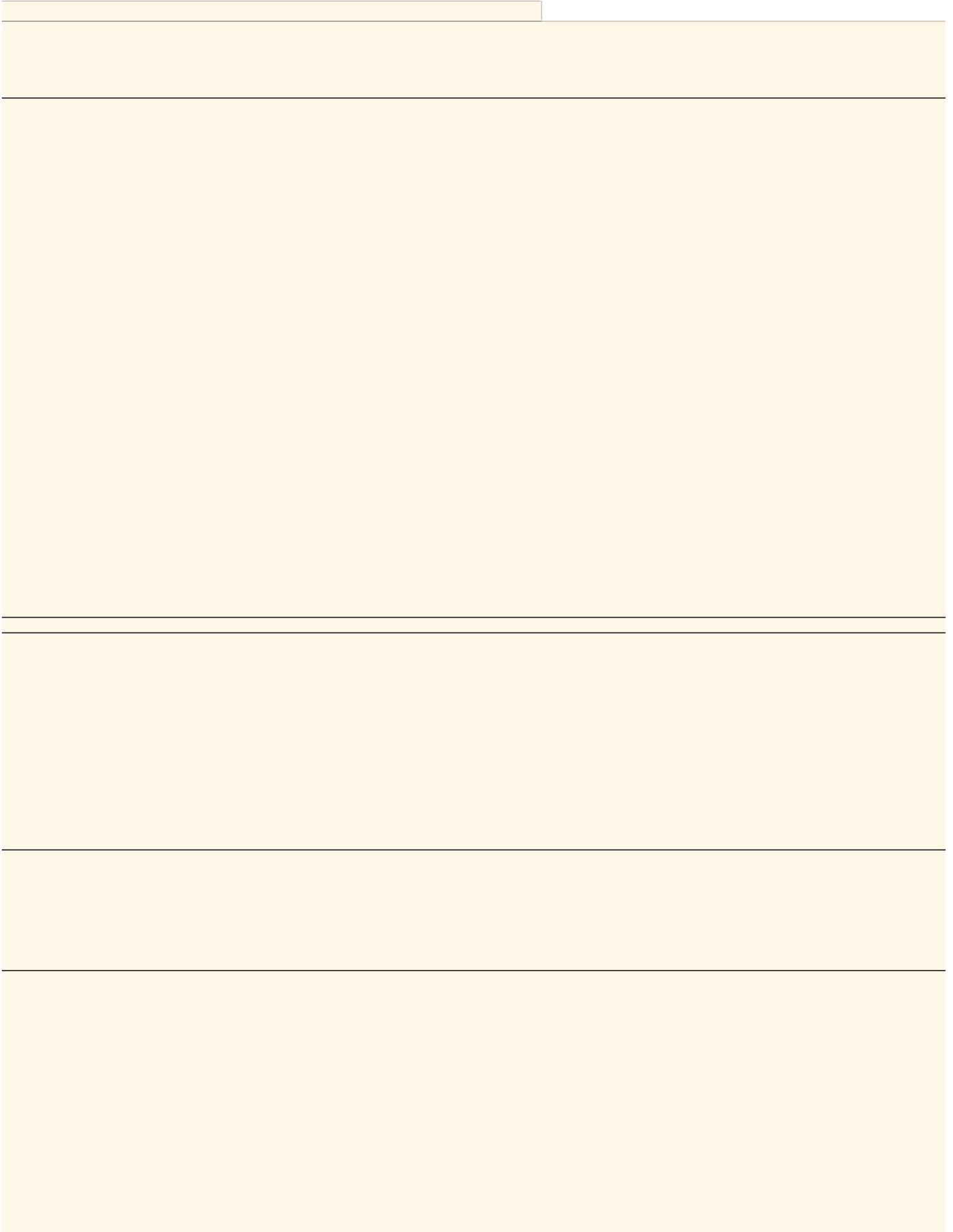
```
XMLManager
+parseBestenliste(bestenliste : String) : BestenlisteManager
+parseEinstellungen(einstellungen : String) : EinstellungenManager
+getBestenlisteXML(bestenliste : BestenlisteManager) : String
+getEinstellungenXML(einstellungen : EinstellungenManager) : String
```

```
ong) : boolean []
```

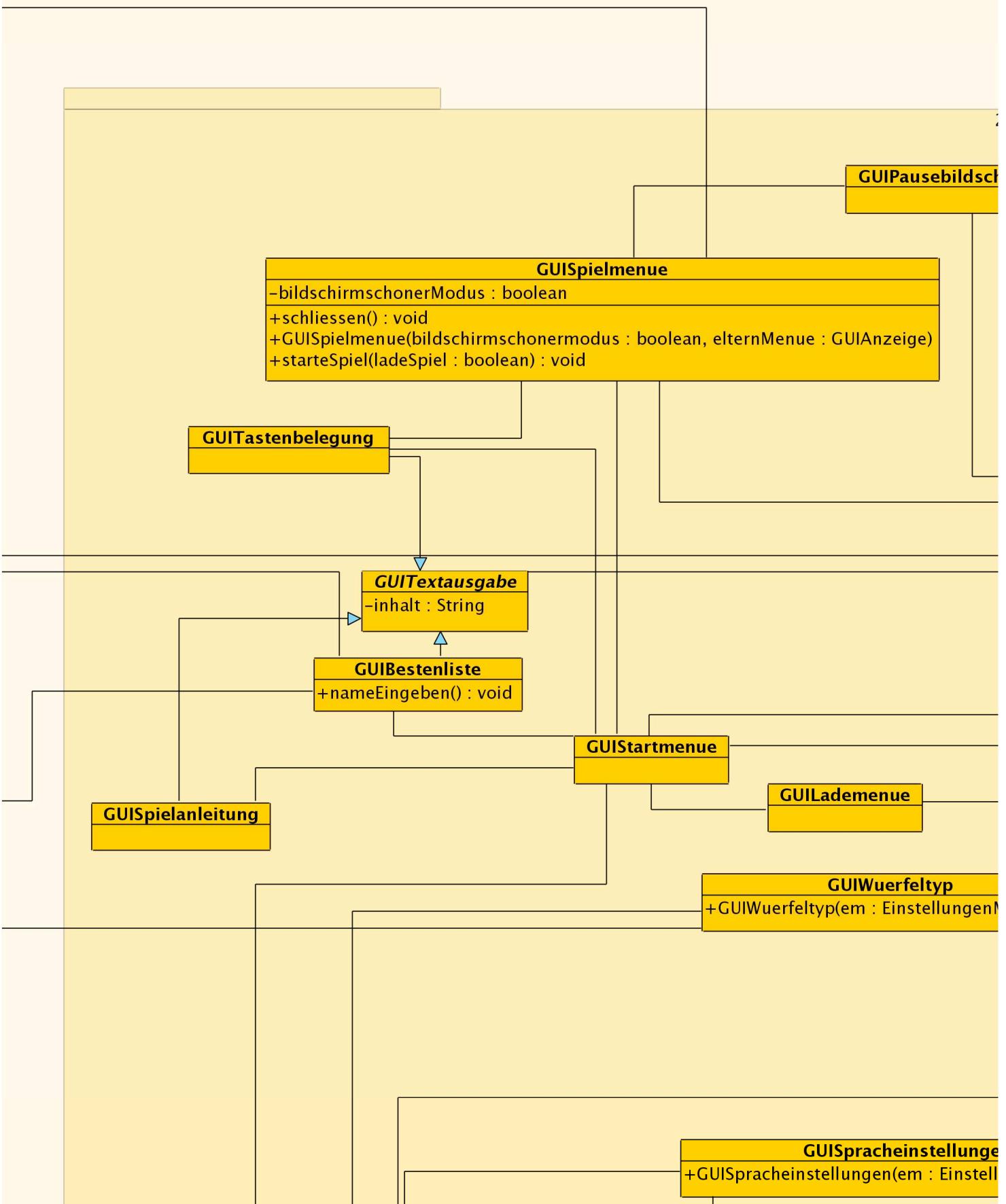
```
SprachManager
+DEUTSCH = 0
+ENGLISCH = 1
+RUMAENISCH = 2
-deutsch = String
-englisch = String
-rumaenisch = String
+getString(key : String) : String
```

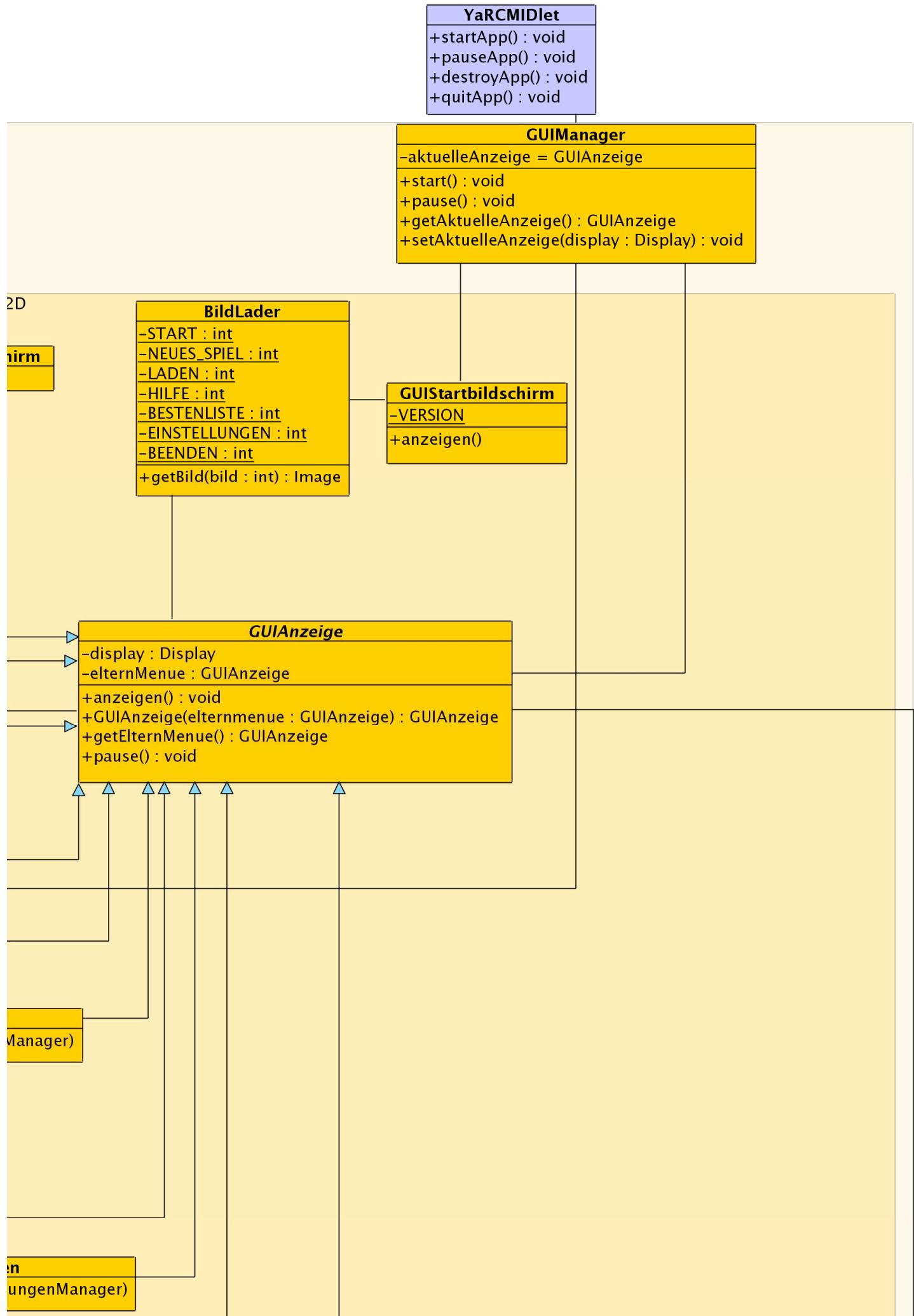
```
SpeicherManager
+getMusikDatei(titelAuswahl : boolean) : InputStream
+getBestenliste() : BestenlisteManager
+speichereBestenliste(bestenliste : BestenlisteManager) : void
+speichereSpiel(spiel : SpielSpeicher) : void
+getSpiel() : SpielSpeicher
+speichereSchwierigkeitsgrad(schwierigkeitsgrad : int) : void
+speichereWuerfel(typ : boolean) : void
+speichereSprache(sprache : int) : void
+speichereLautstaerke(lautstaerke : int) : void
+getEinstellungen() : EinstellungenManager
+speichereEinstellungen(einstellungen : EinstellugnenManager...)
```

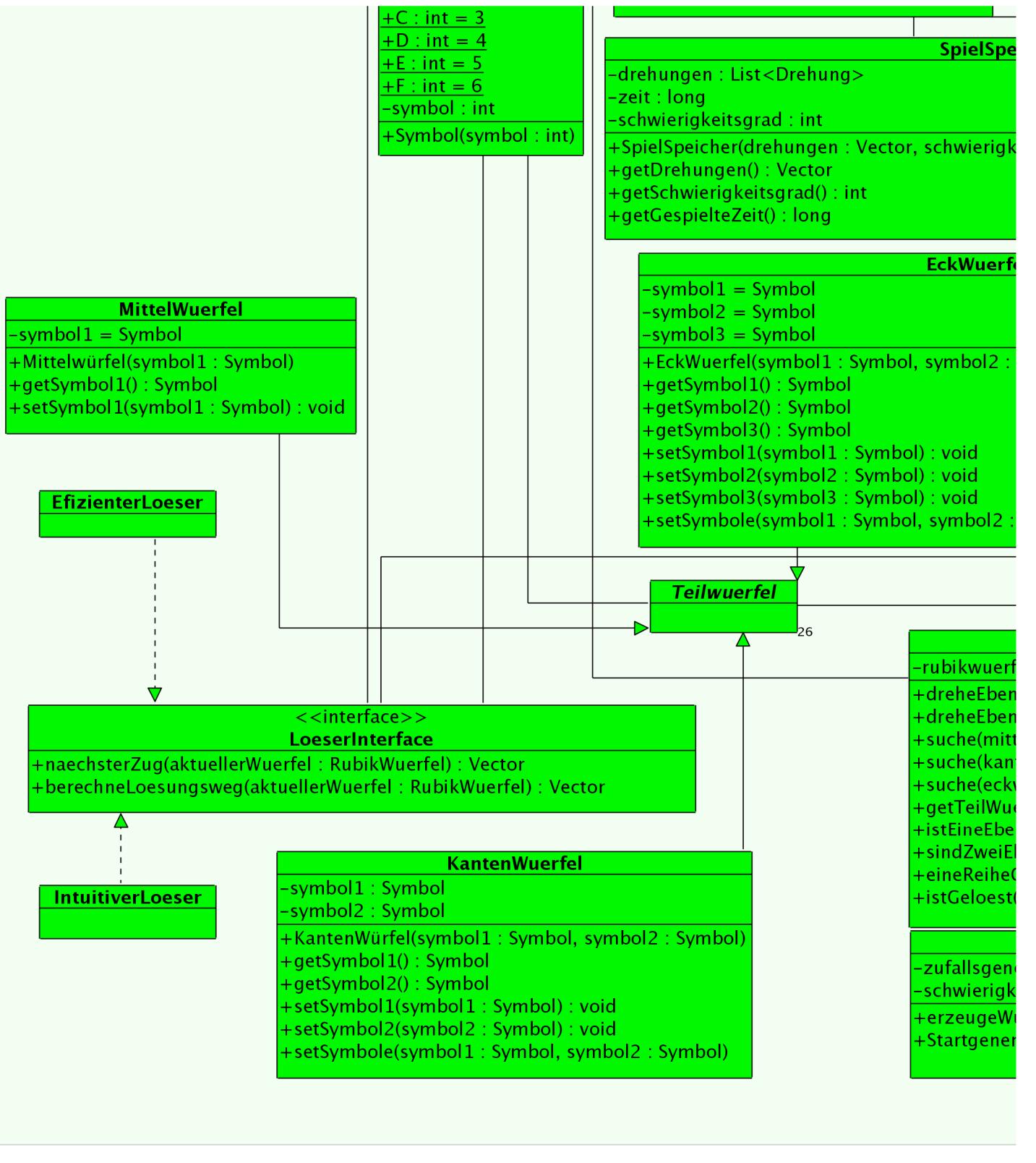
```
ZeitManager
-spielzeit : long
-startZeit : long
-stopZeit : long
+run() : void
+pause(p : boolean)
```

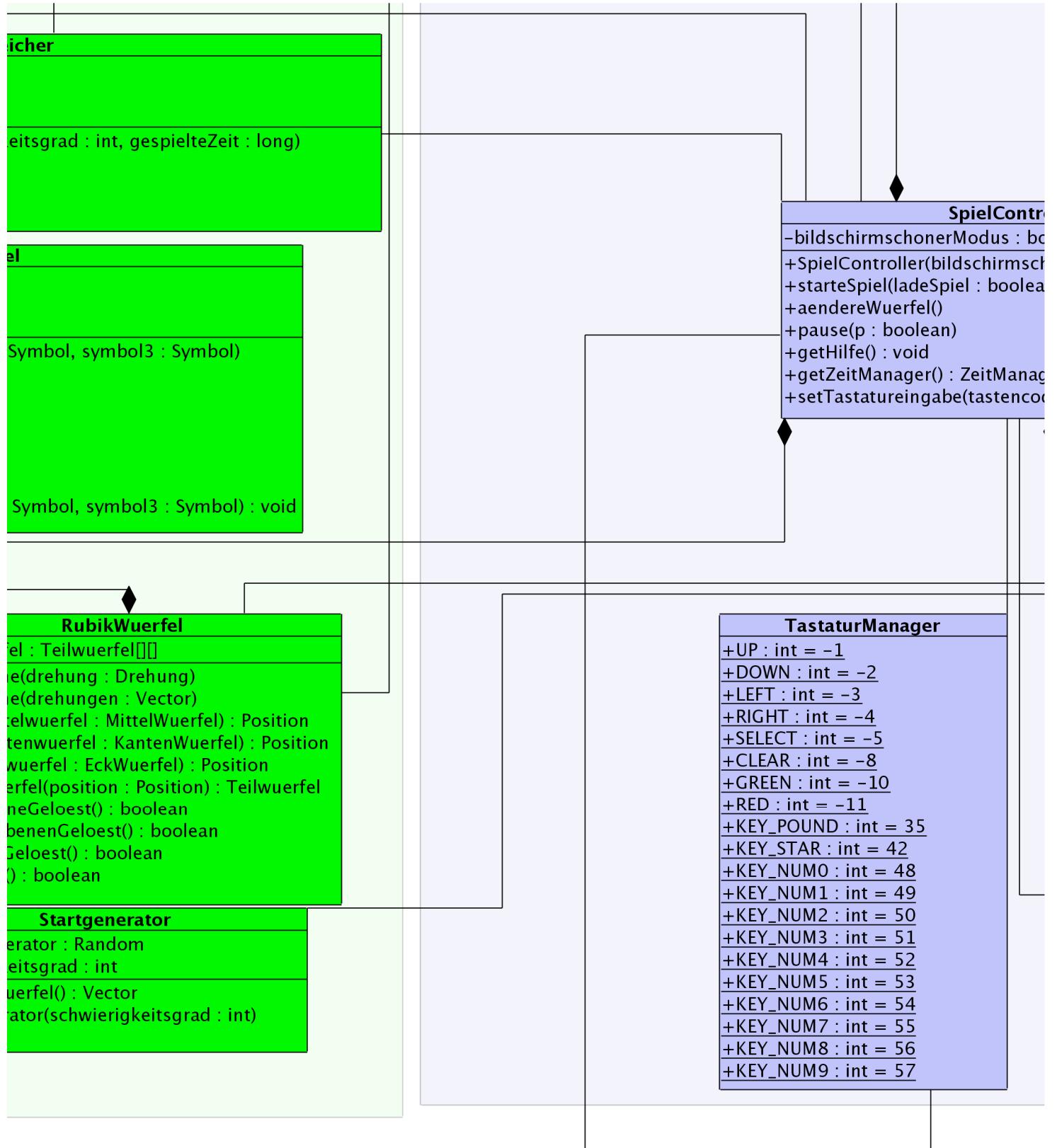


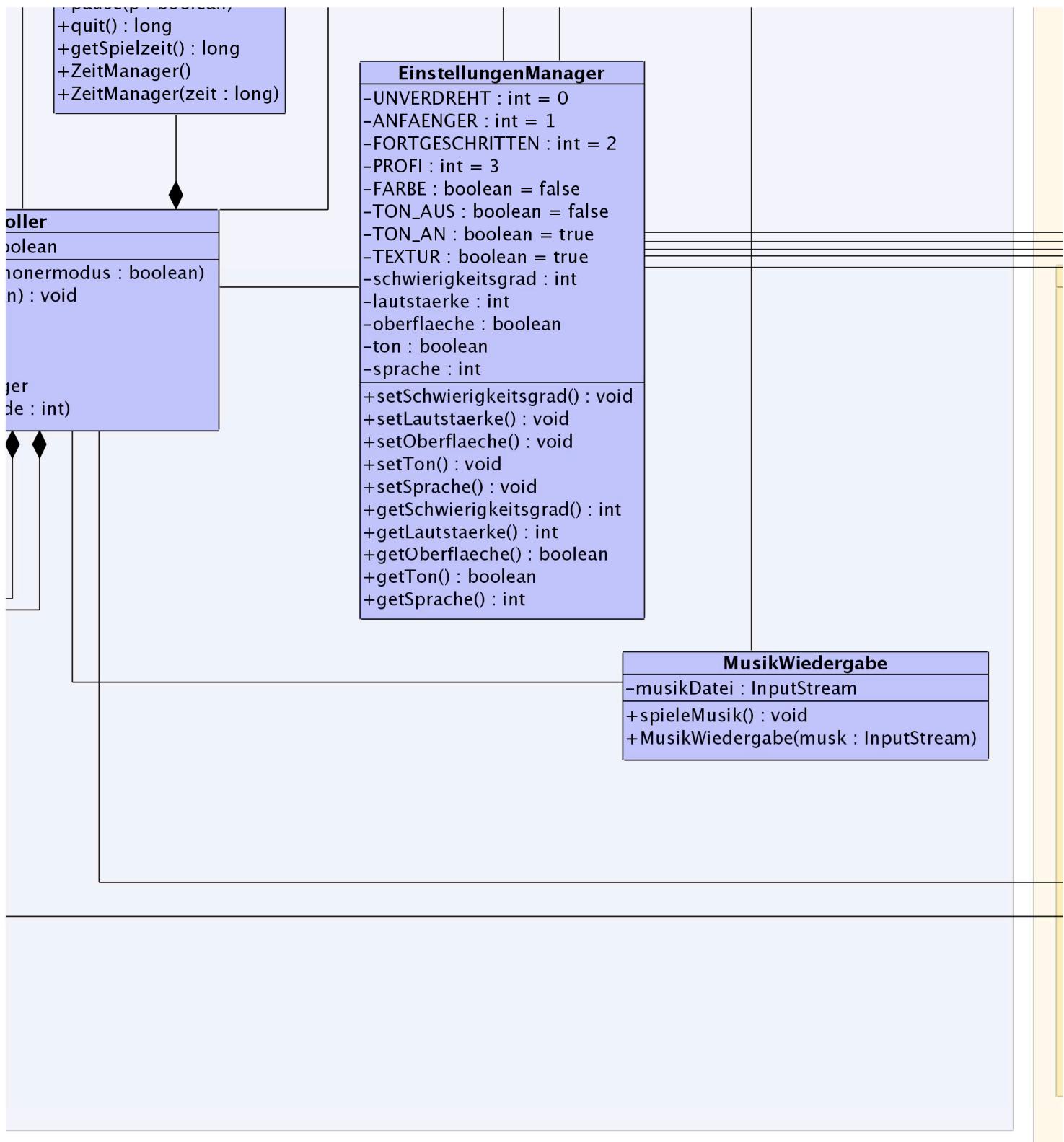
view

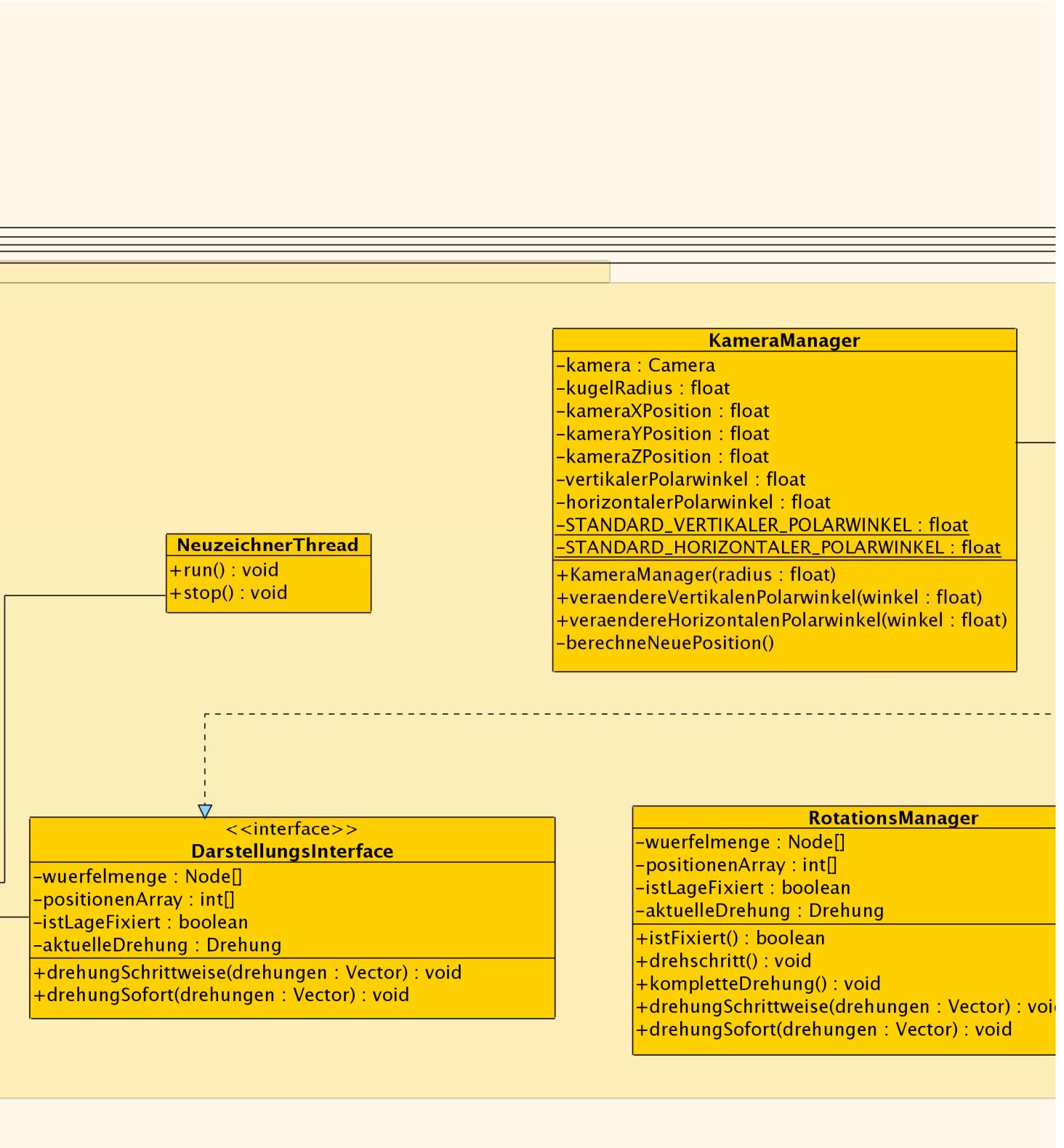


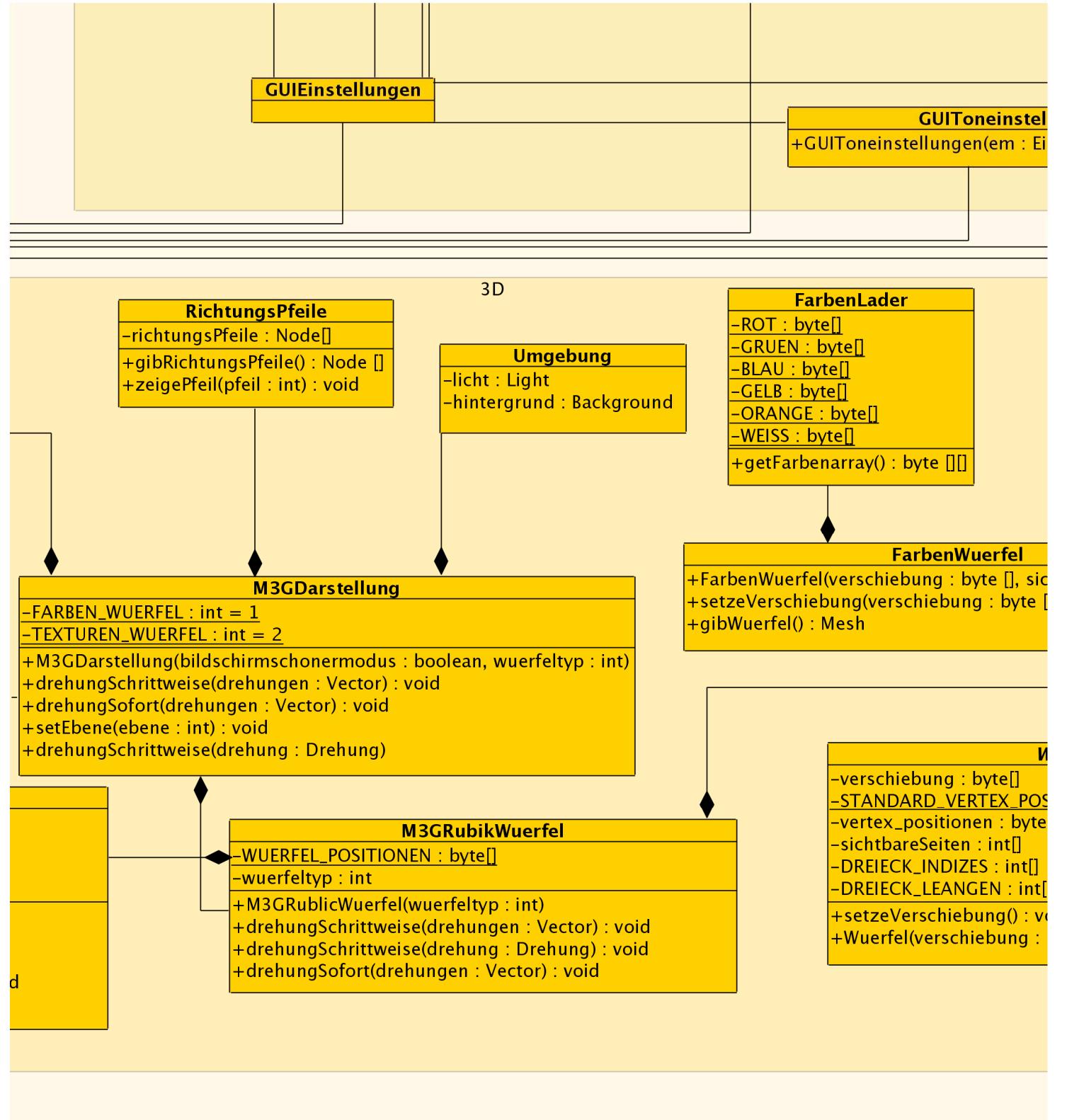


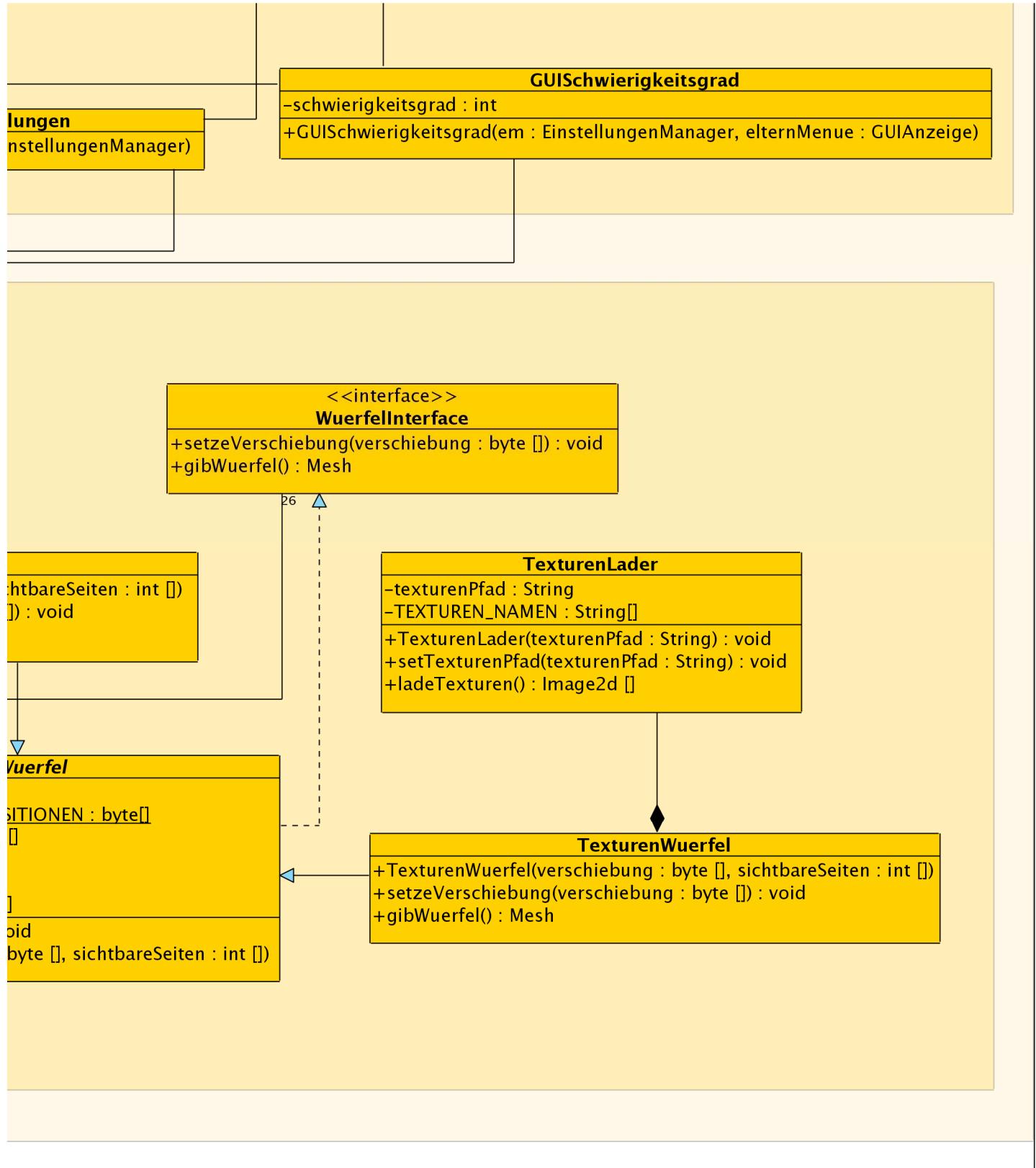












Glossar

- LCD Ein Flüssigkristallbildschirm (englisch liquid crystal display, kurz LCD), ist ein Bildschirm, bei dem spezielle Flüssigkristalle zur Bilddarstellung genutzt werden, die die Polarisationsrichtung von Licht beeinflussen können. Diese Bildschirme, deren Realisierung nur mit einer Matrix von Dünnenschichttransistoren (TFT: thin-film transistor) möglich ist, stellen die zurzeit dominante Flachbildschirm-Technologie dar., [4](#)
- MVC siehe [Model-View-Controller-Prinzip](#), [4](#)
- redo Bezeichnet die Möglichkeit, nach einem Rückgängigmachen (siehe [undo](#)) dieselben Schritte ohne Neueingabe nochmals ausführen zu lassen., [4](#)
- undo (englisch für rückgängig machen) Ist die Bezeichnung für die Funktion von Computerprogrammen, eine oder mehrere Eingaben zurückzunehmen. Je nach Komplexität der Anwendung findet sich keine Undo-Möglichkeit oder ein nur einstufiges Undo bis hin zur Rücknahme sämtlicher Arbeitsschritte., [4](#)