Projektarbeit:

KB-TRIS – Ein Tetris-Clone

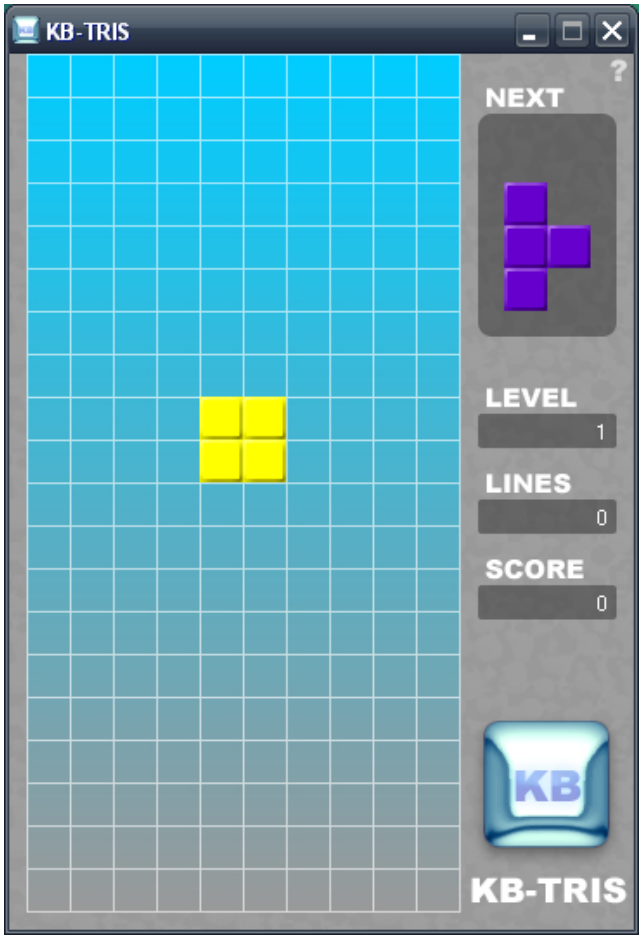
Eingereicht von: Hermann, Kai
Halbrock, Benjamin

Bearbeitungszeit: von 14.03.08
bis 19.06.08
(ca. 3 Monate)

HEINRICH-WIELAND-SCHULE
PFORZHEIM
Technisches Gymnasium Fachrichtung Informationstechnik

Projektdokumentation KB-TRIS (Tetris)

Menü



Im Spiel

Abstrakt

Das Projekt KB-TRIS ist eine Umsetzung des Spielprinzips von Tetris in C++.

Es wurde das Fallen und Aufsetzen der 7 verschiedenen Steine realisiert, ebenso wie das Drehen und Verschieben jener.

Des weiteren wurde eine Pausefunktion und eine Highscoreverwaltung eingebaut.

Das Fallen der Steine beschleunigt sich nach einer bestimmten Anzahl gelöschter Reihen.
Dies wird durch die Level-Anzeige sichtbar gemacht.

Auch die Anzahl der gelöschten Reihen wird angezeigt.

Inhaltsverzeichnis

Anleitung bzw. Hilfe

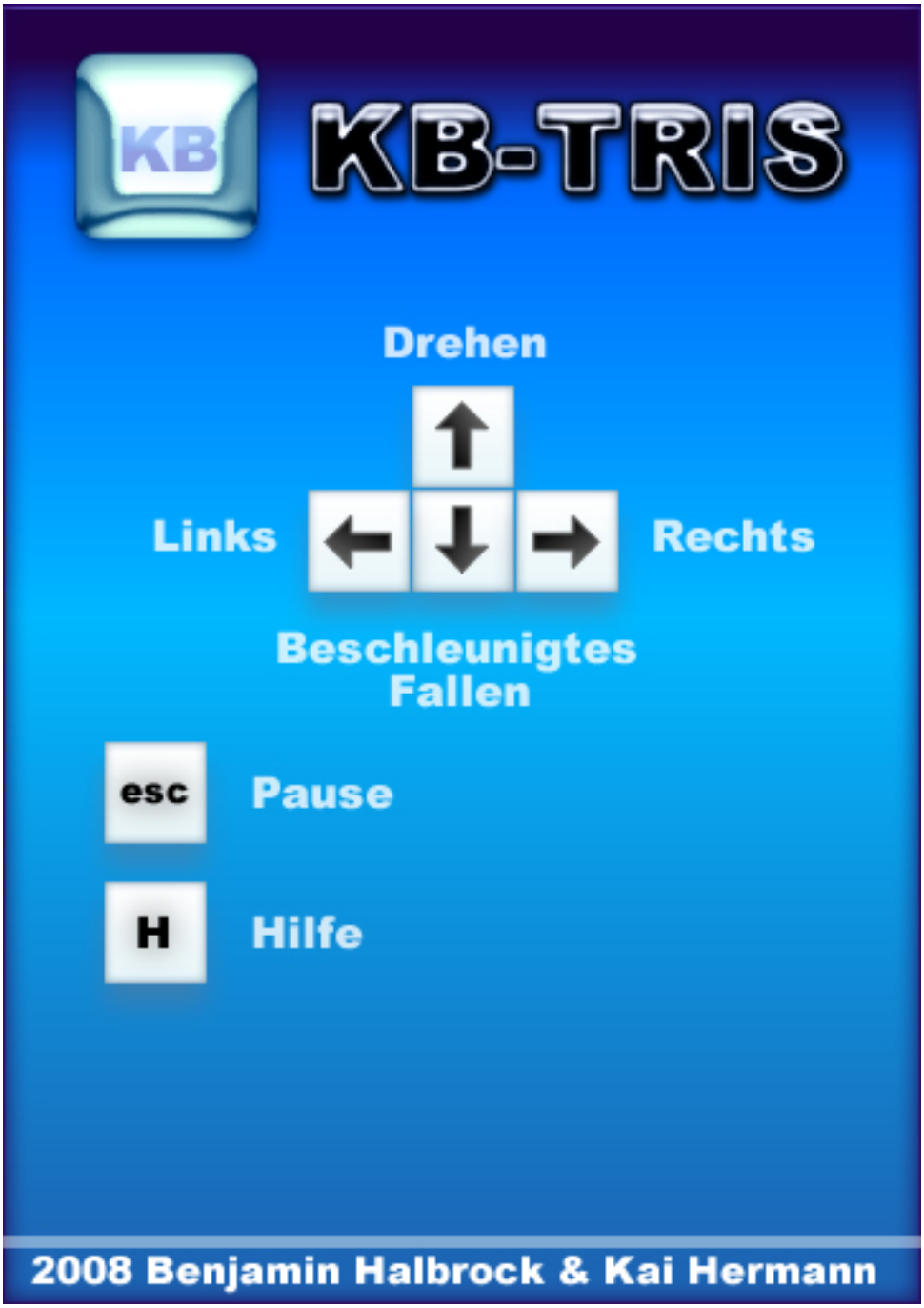
Ziel bei KB-TRIS ist es die von oben herunterfallenden Steine durch geschicktes Drehen so zu stapeln, dass sich geschlossene Reihen bilden.

Eine solche Reihe verschwindet dann und das ganze Spielfeld sinkt ab.

Für jede volle Reihe gibt es Punkte.

Der fallende Stein wird über die Pfeiltasten gesteuert.

Durch drücken von „H“ ruft man die Hilfe auf.
Drückt man „Escape“, so wechselt das Spiel in den Pause-Modus.



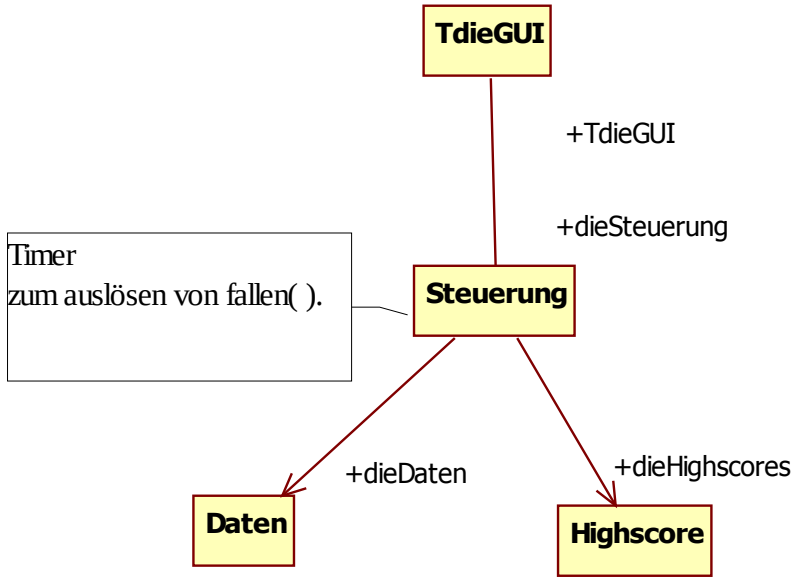
Einleitung

KB-TRIS beruht auf der 3-Schichtenarchitektur.

Die Benutzerschnittstelle

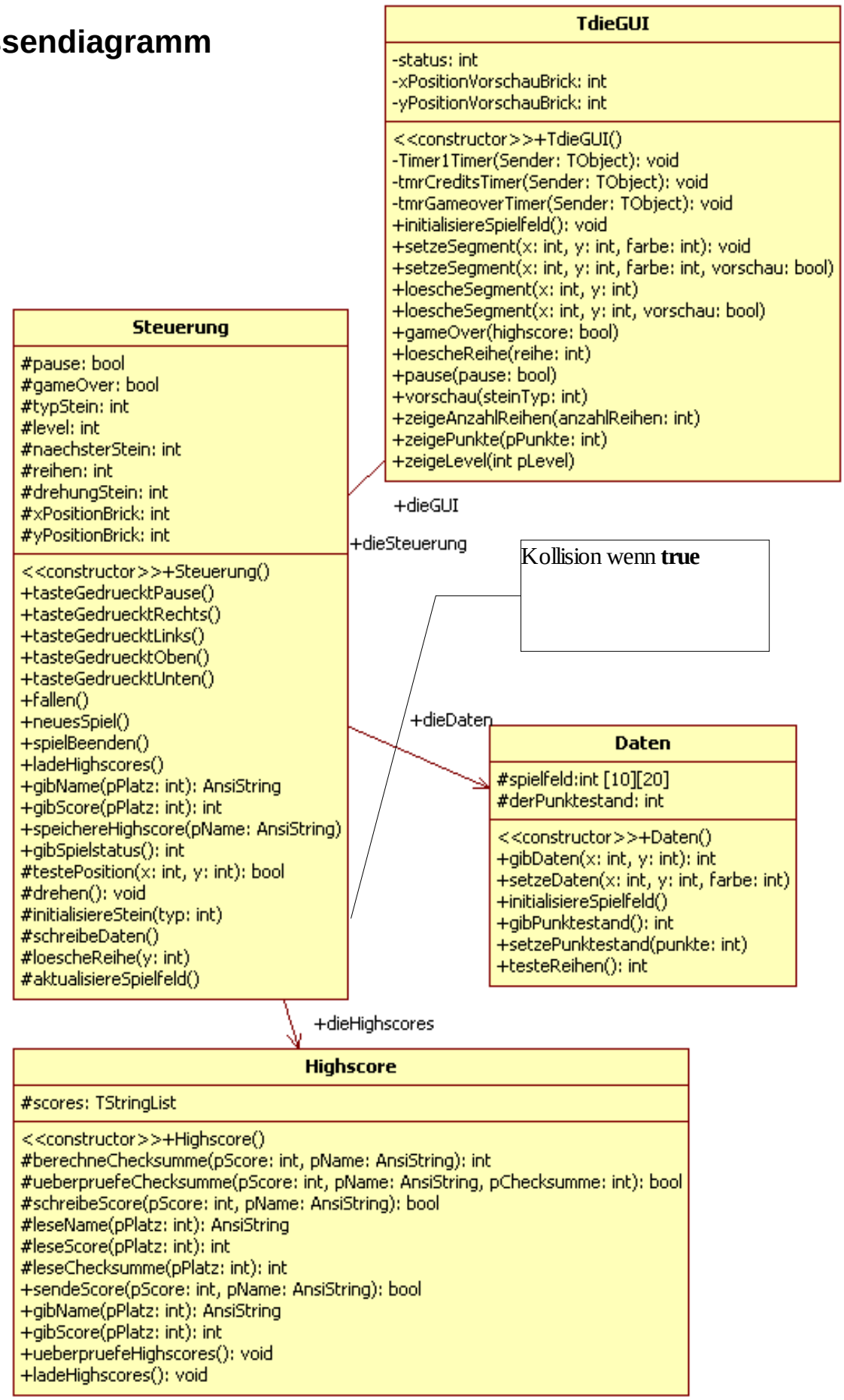
Der Kern der Anwendung

Die Datenhaltungsschicht



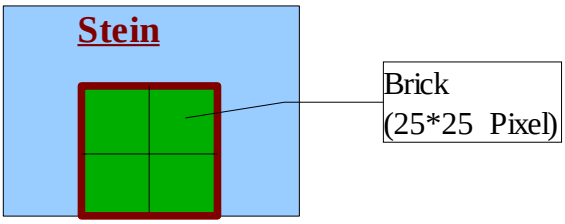
Da wir den Borland C++ Builder verwendet haben, sind die Timer in die GUI integriert.
Und die Abfrage der Pfeiltasten wurde durch einen kleinen Hack realisiert.
Siehe **Eingabebuffer.**

Klassendiagramm

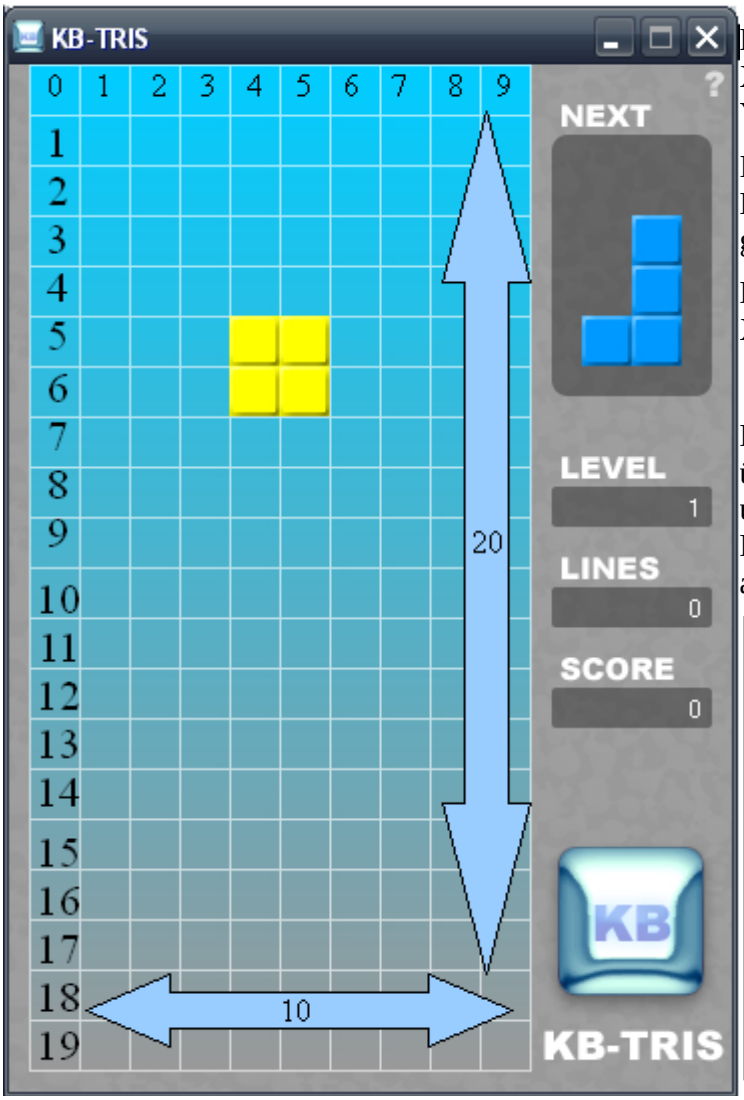


Beschreibung des Spielsteins

Ein fallender Stein besteht aus 4-Bricks.
Deren Position wird in den Feldvariablen xPositionBrick:int[4] und yPositionBrick:int[4] der Steuerung gespeichert.
Beim ändern der Position des Steins, werden alle Bricks verschoben.



Beschreibung des Spielfeldes



Das Spielfeld besteht aus 10 Feldern in X-Richtung und 20 Feldern in Y-Richtung.

Im Attribut spielfeld:int[10][20] der Klasse Daten wird das Spielfeld gespeichert.

Der erste Teil der Variablen beschreibt die X-Werte, der zweite Teil die Y-Werte.

Die folgende Tabelle gibt einen Überblick über die Zuordnung der Werte der Daten und der ausgegebenen Farbe.
Durch ändern der Grafiken lässt sich das aber ändern.

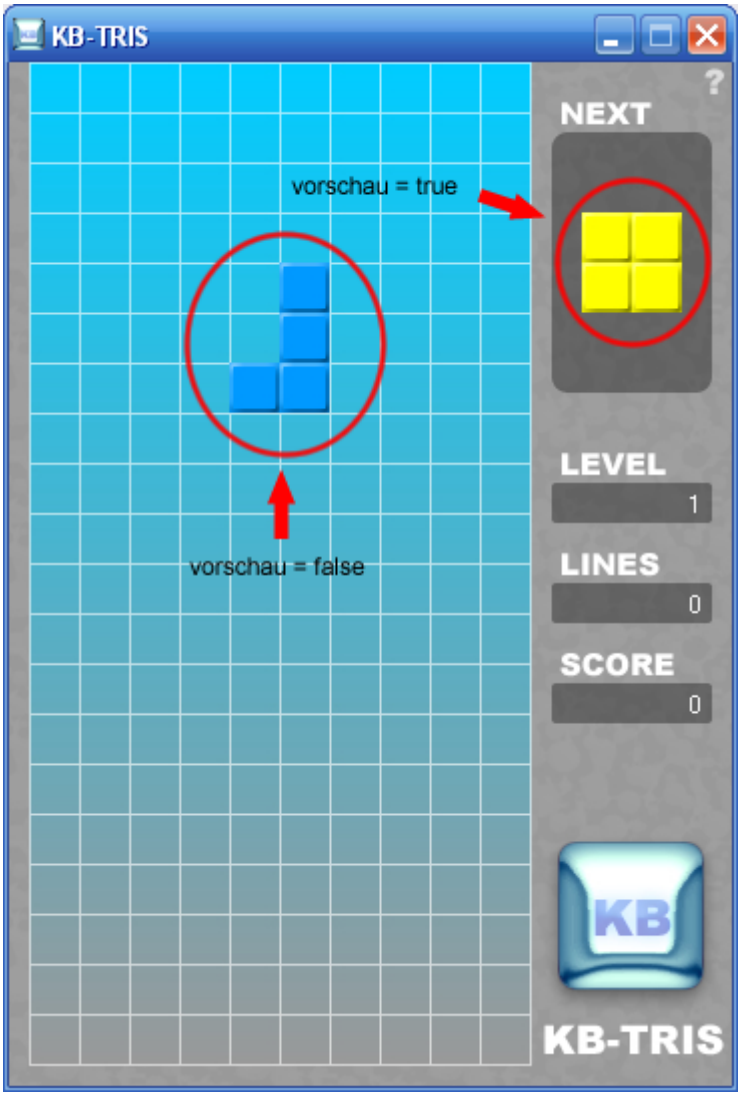
Wert	Farbe
-1	leer
0	hellblau
1	blau
2	orange
3	gelb
4	grün
5	lila
6	rot

Beschreibung einzelner Funktionen

TdieGUI:

Die Operation setzeSegment(int x, int y, int farbe, bool vorschau) zeigt ein Segment auf einem der beiden TImage Felder der GUI an.
Ist vorschau **true** wird das Vorschaufeld "TImage *vorschaufeld[2][4]" angesteuert,
wenn vorschau **false** wird das Spielfeld "TImage *spielfeld[10][20]" angesteuert.

Die Segmentfarbe wird durch den Parameter "farbe" festgelegt, die Zuordnung der Zahlenwerte zu den entsprechenden Farben erfolgt wie unter "Beschreibung des Spielfeldes" erläutert.



Steuerung:

Die Steuerung wird im Konstruktor der GUI erstellt.
Der Konstruktor der Steuerung erzeugt die Daten und die Highscoreverwaltung.

#pause:bool

Die Variable wird mit **true** initialisiert.

Beim starten eines neuen Spiels [neuesSpiel( )] wird sie auf **false** gesetzt.

Beträgt ihr Wert **true**, werden die Operationen tasteGedruecktRechts( ), tasteGedruecktLinks( ), fallen( ) und drehen( ) übersprungen.

Die Operation tasteGedruecktPause( ) invertiert ihren Wert. (ESCAPE)

#gameOver:bool

Eigentlich gleich wie **pause:bool.**
Es wird jedoch zusätzlich die Operation tasteGedruecktPause( ) gesperrt.

Sie wird **true**,wenn die Operation spielBeenden( ) ausgeführt wird.

#int level

Diese Variable beeinflusst das Intervall des Timers.

Daten:

Im Attribut **spielfeld:int[10][20]** wird das Spielfeld gespeichert.

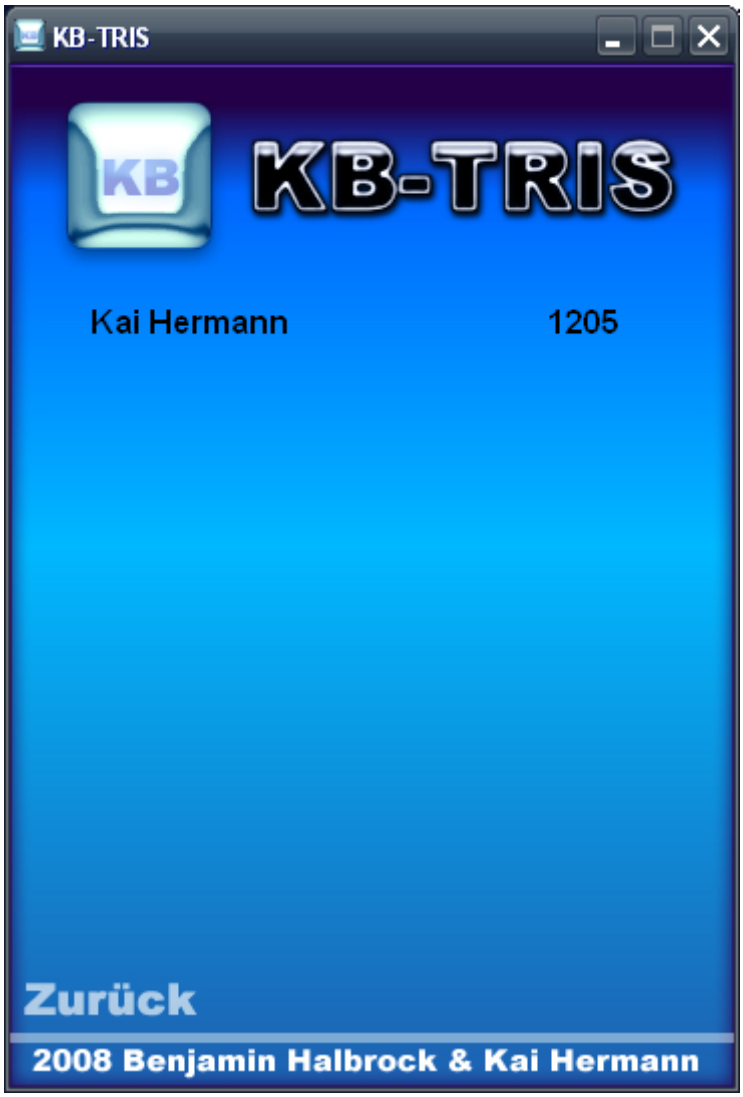
Der erste Teil der Variablen beschreibt die X-Werte, der zweite Teil die Y-Werte.

Die folgende Tabelle gibt einen Überblick über die Zuordnung der Werte der Daten und der zugeordneten Farbe.
Durch ändern der Grafiken lässt sich das aber ändern.

Wert	Farbe
-1	leer
0	hellblau
1	blau
2	orange
3	gelb
4	grün
5	lila
6	rot

Highscore:

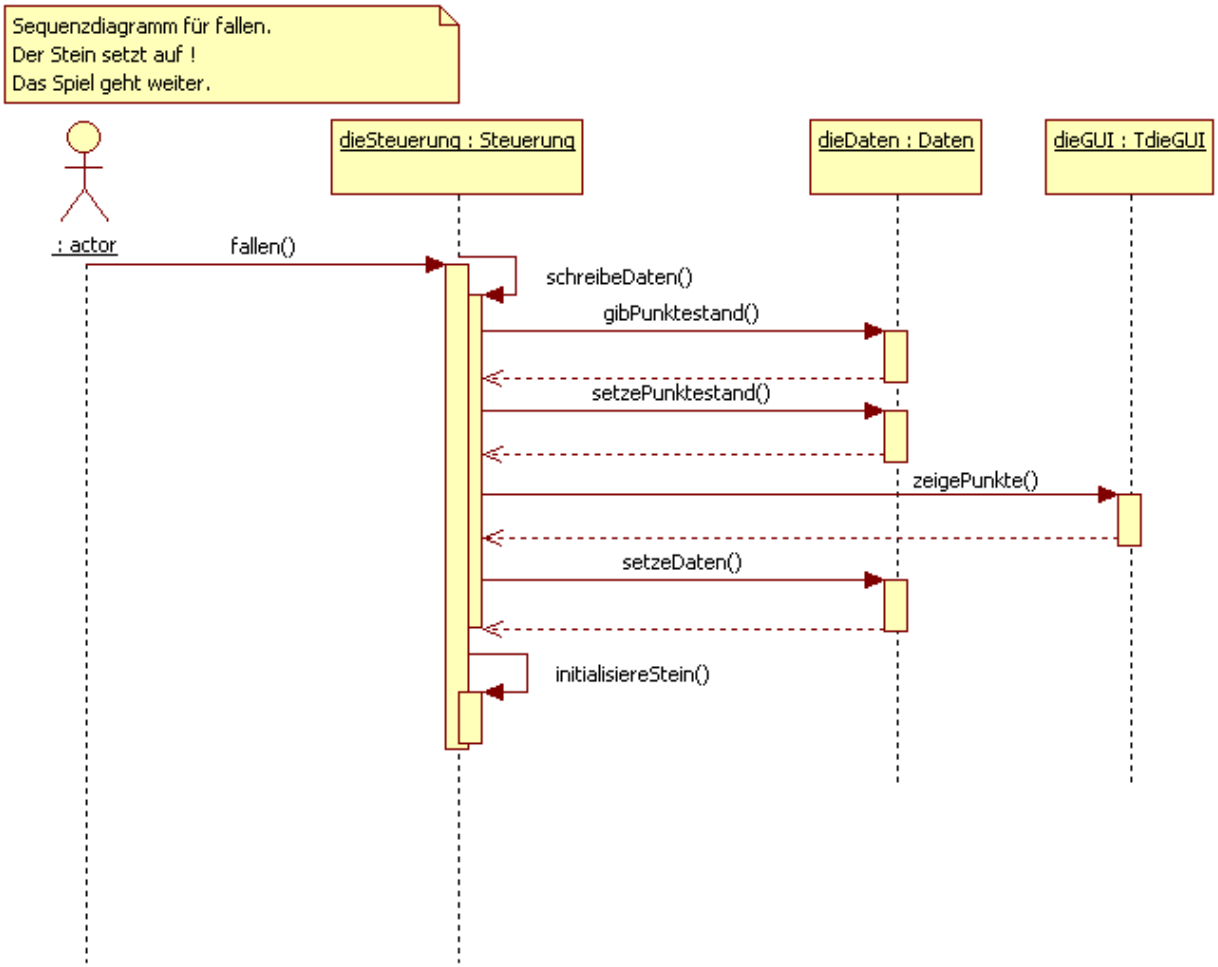
Da die Highscoreverwaltung eine gewisse Intelligenz besitzt, wurde sie nicht in die Daten integriert.
Sie ließt ihre Werte bei Programmstart ein und speichert sie auch wieder in einer Datei ab.
Zu Name und Punktzahl wird auch eine Prüfsumme berechnet, was Tricksen verhindern soll.



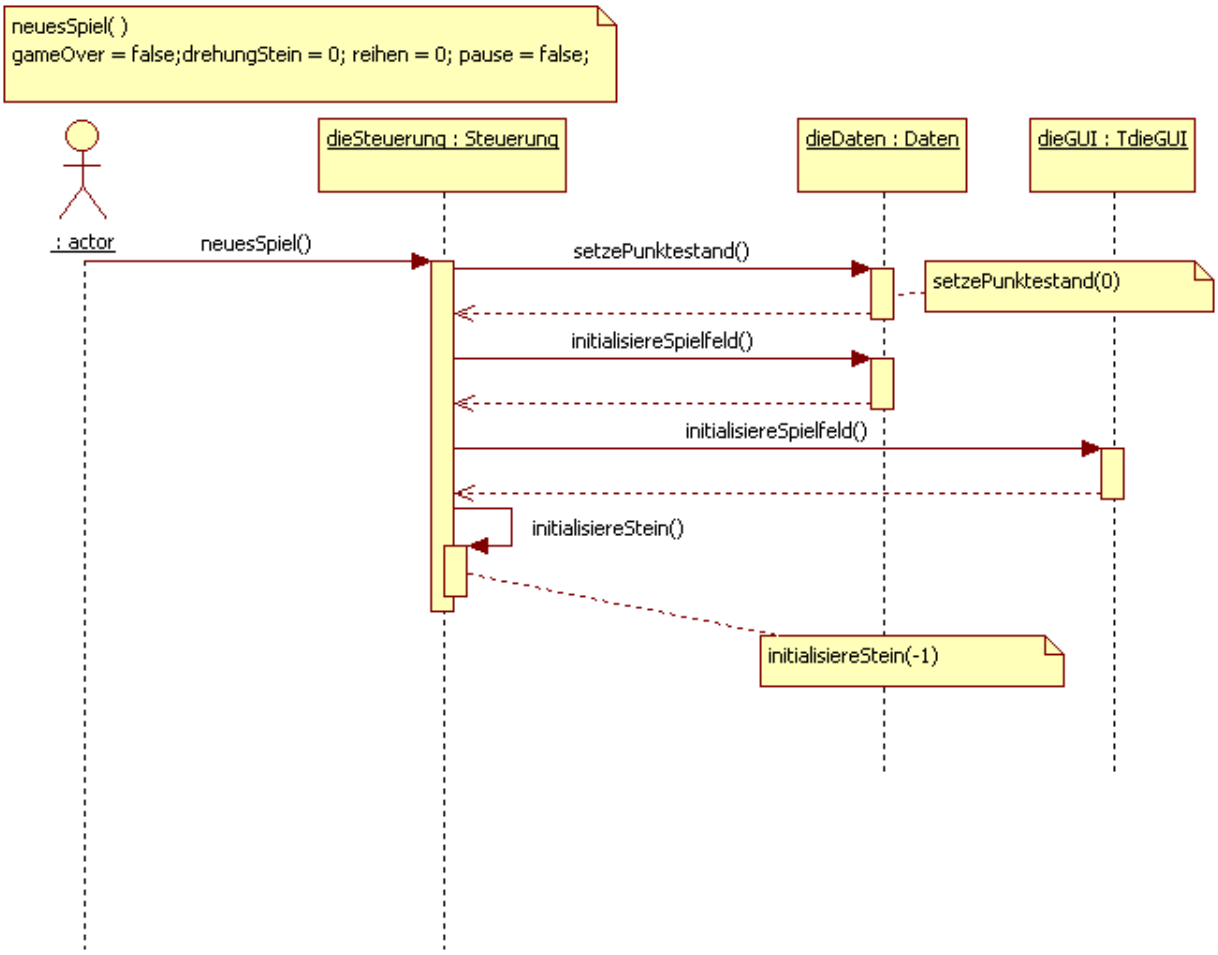
Die Highscoreanzeige

Sequenzdiagramme

Steuerung::fallen, Stein setzt auf, Spiel geht weiter



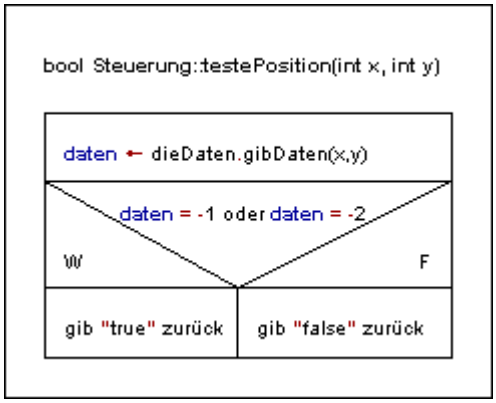
Steuerung::neuesSpiel



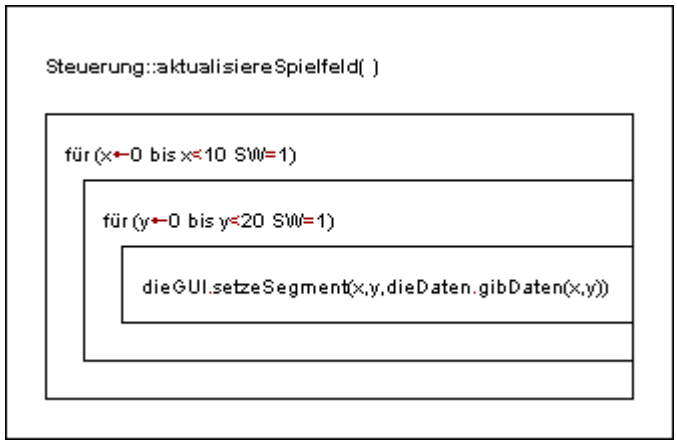
Struktogramme

Operation testePosition(int x, int y) der Steuerung

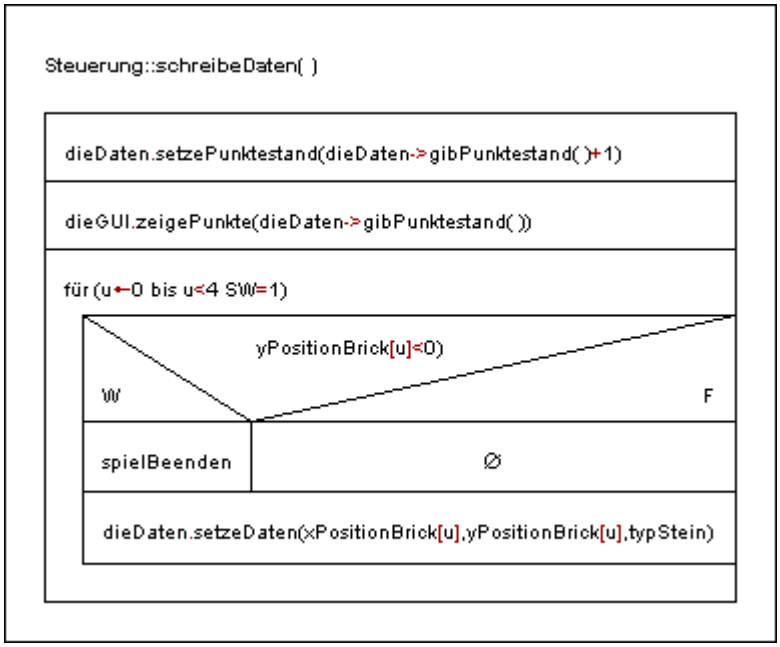
Ist der Rückgabewert **true**, so findet eine Kollision statt.



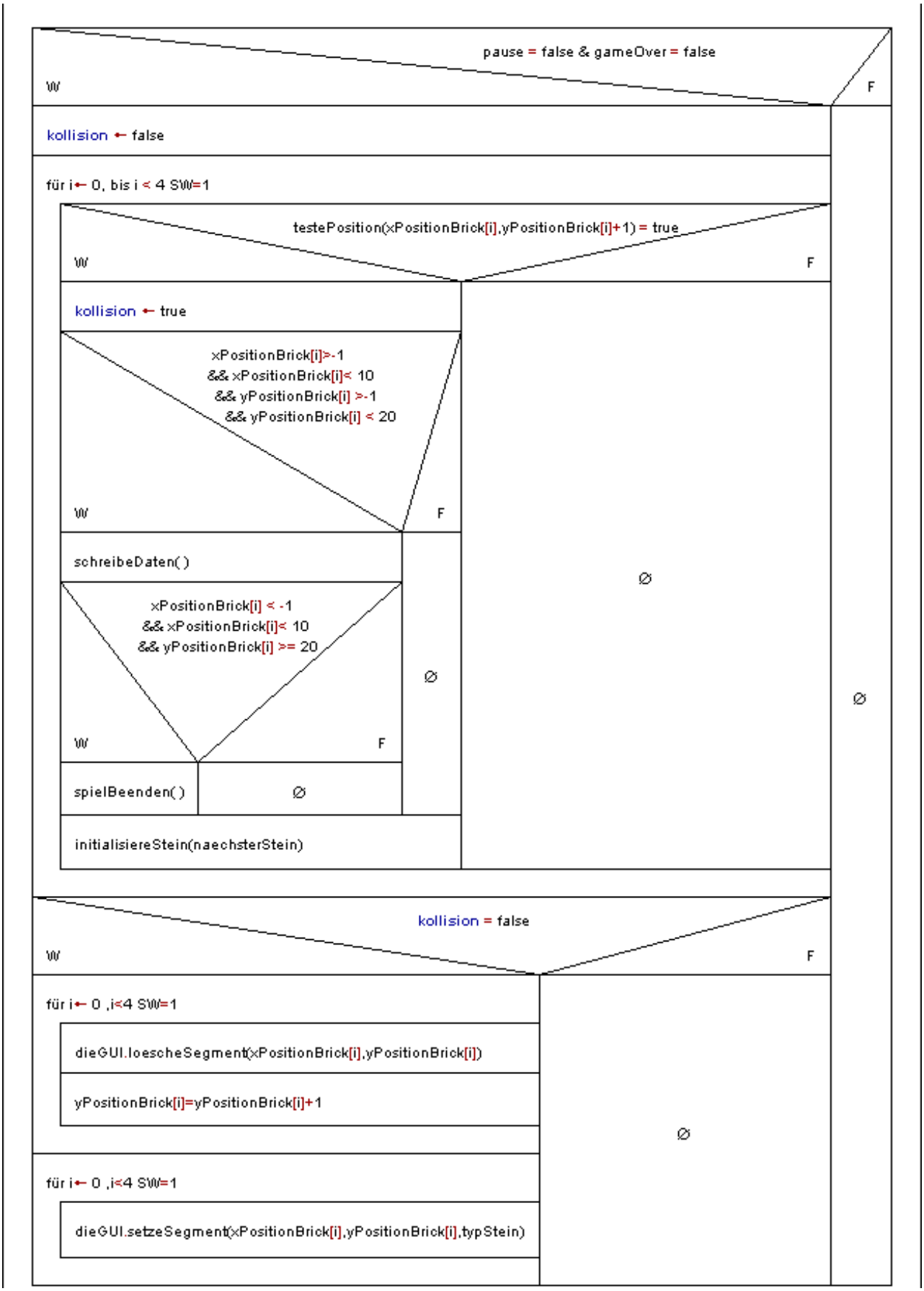
Operation aktualisiereSpielfeld( ) der Steuerung



Operation schreibeDaten( ) der Steuerung



Operation fallen( ) der Steuerung

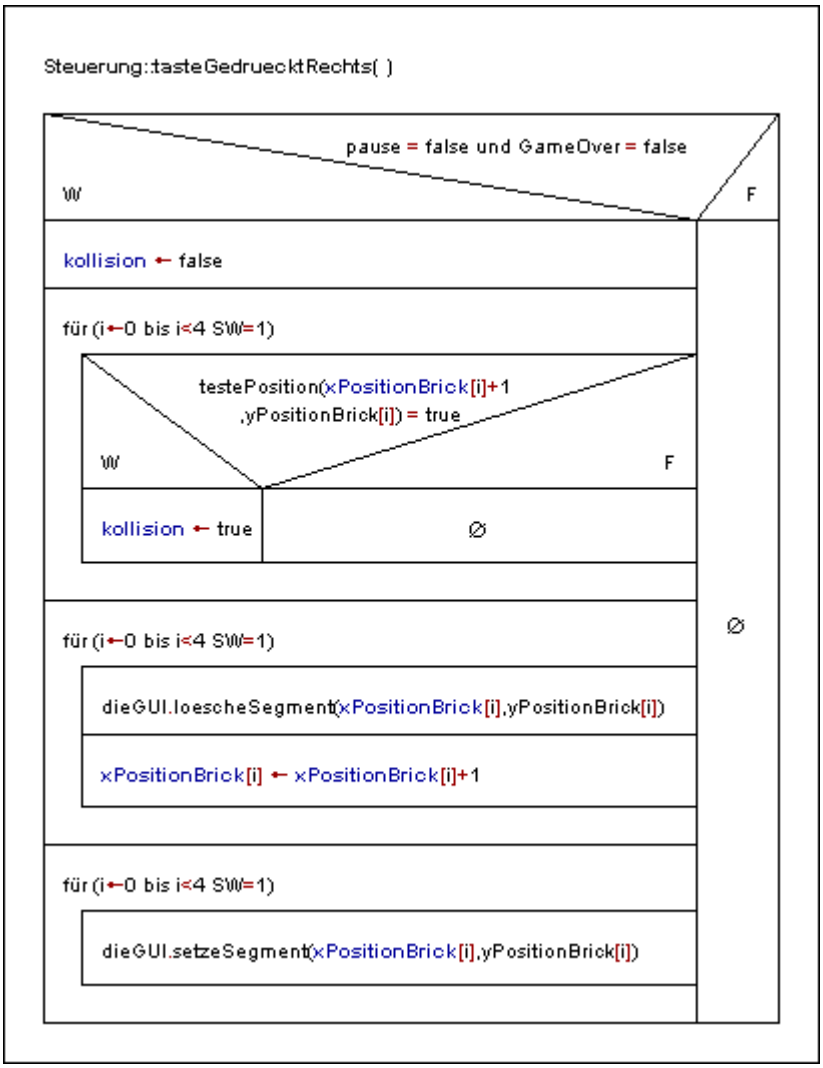


Operation tasteGedruecktRechts( ) der Steuerung

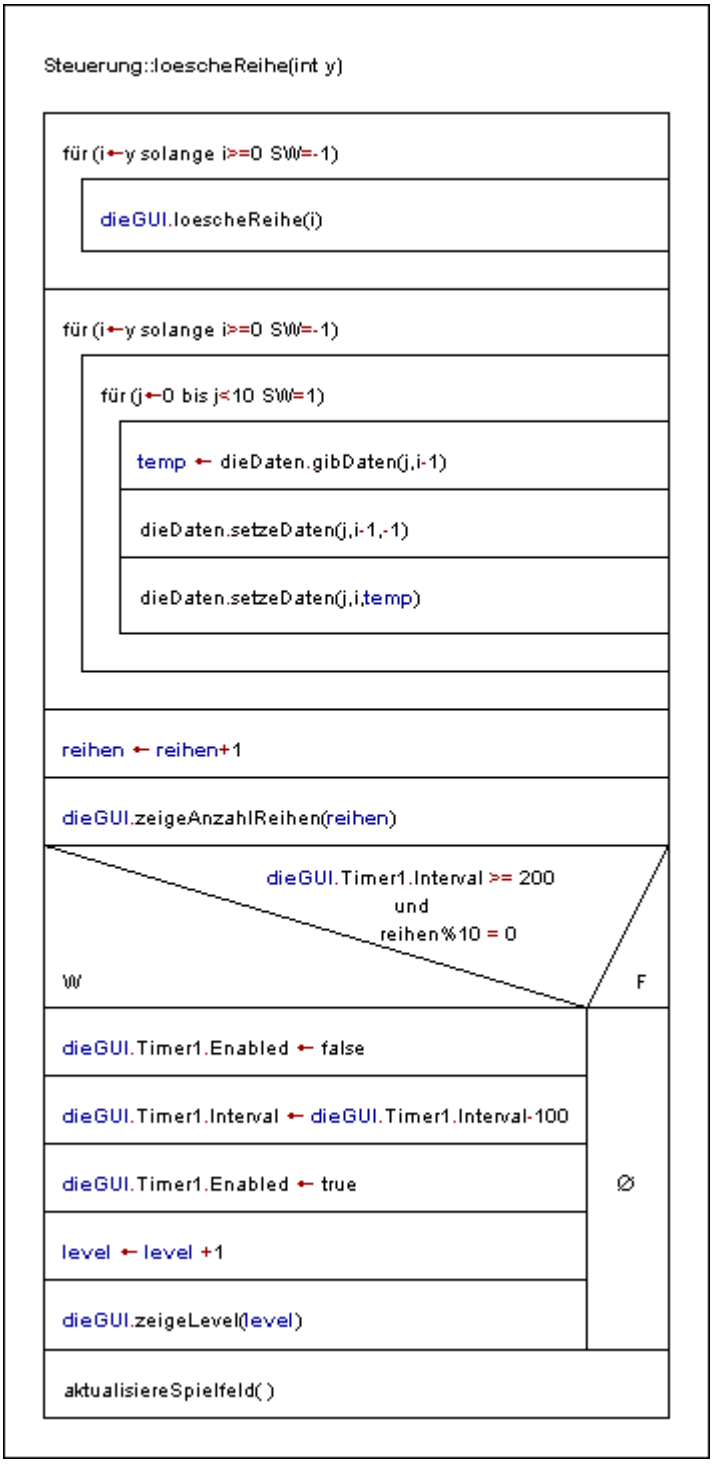
Anmerkung: tasteGedrücktLinks( ) ist im Prinzip gleich aufgebaut.

Es wird für jeden Brick getestet, ob eine Kollision mit der Wand oder einem Stein stattfindet.

Wenn keine stattfindet, werden alle auf der GUI angezeigten Bricks gelöscht, ihre Positionsvariablen geändert und dann alle wieder auf der GUI angezeigt.



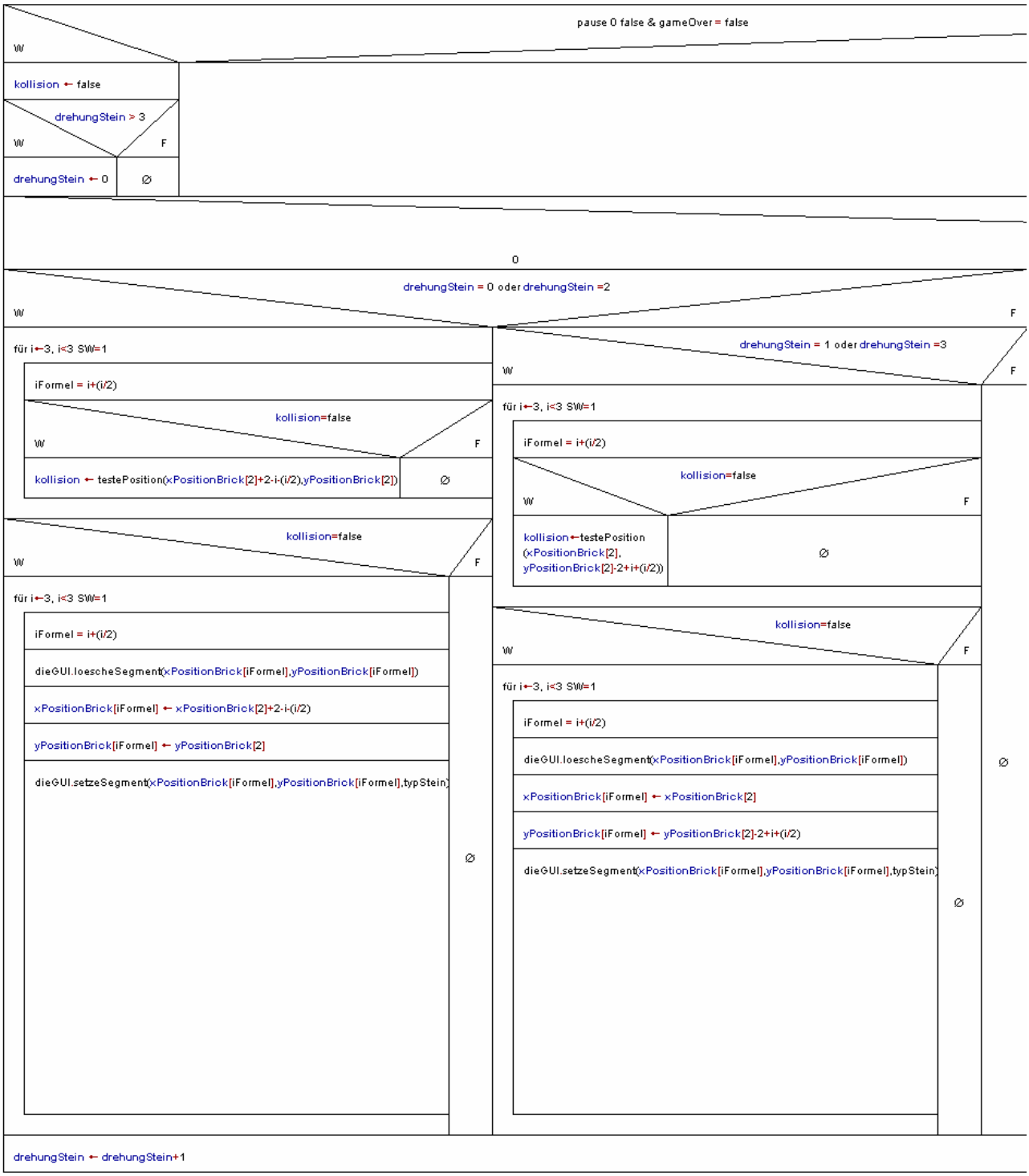
Operation loescheReihe(int y)



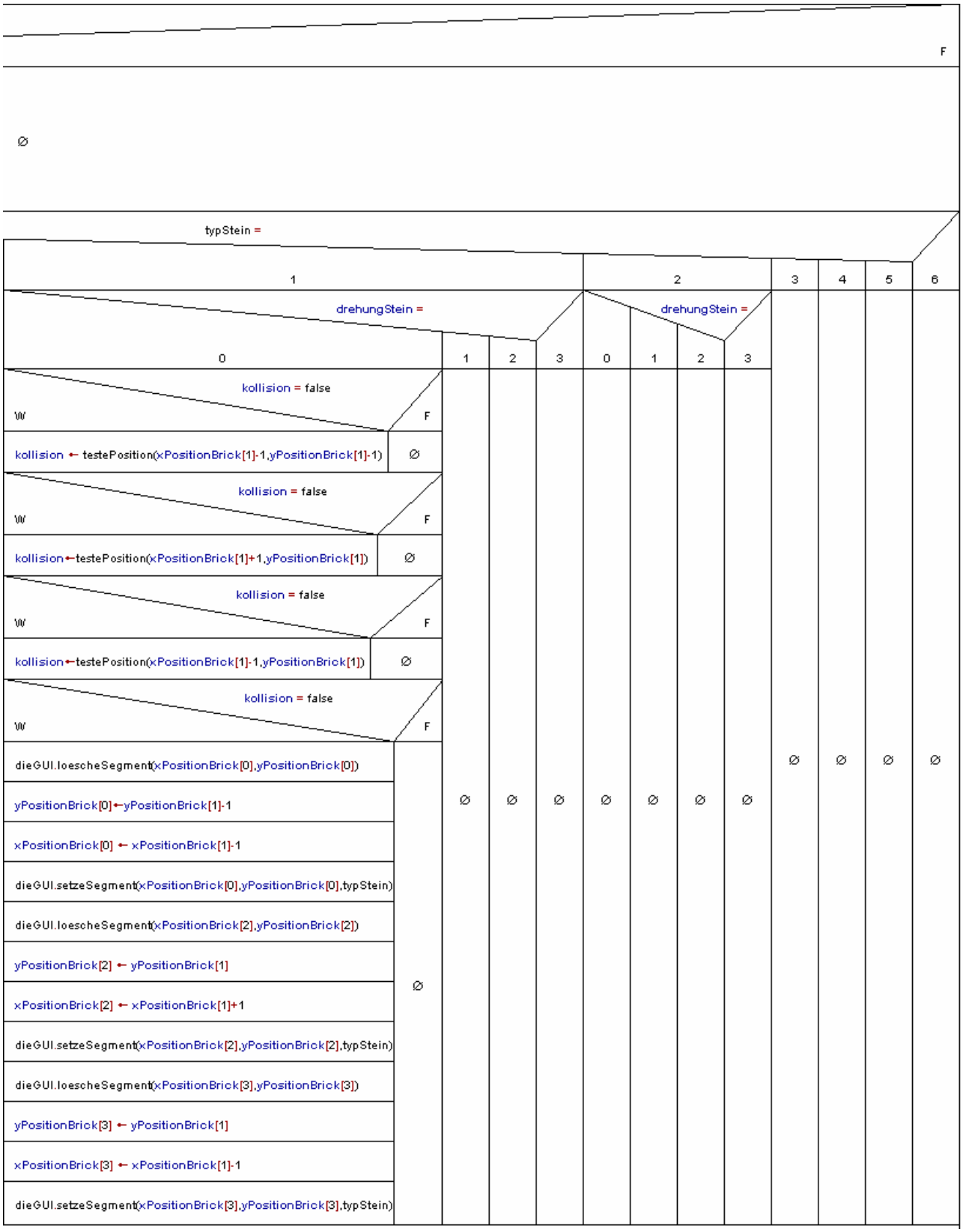
Operation drehen( ) der Steuerung

Aus Platzgründen hier in Kurzform für ausführliche Ansicht siehe Anhang.

Teil 1:

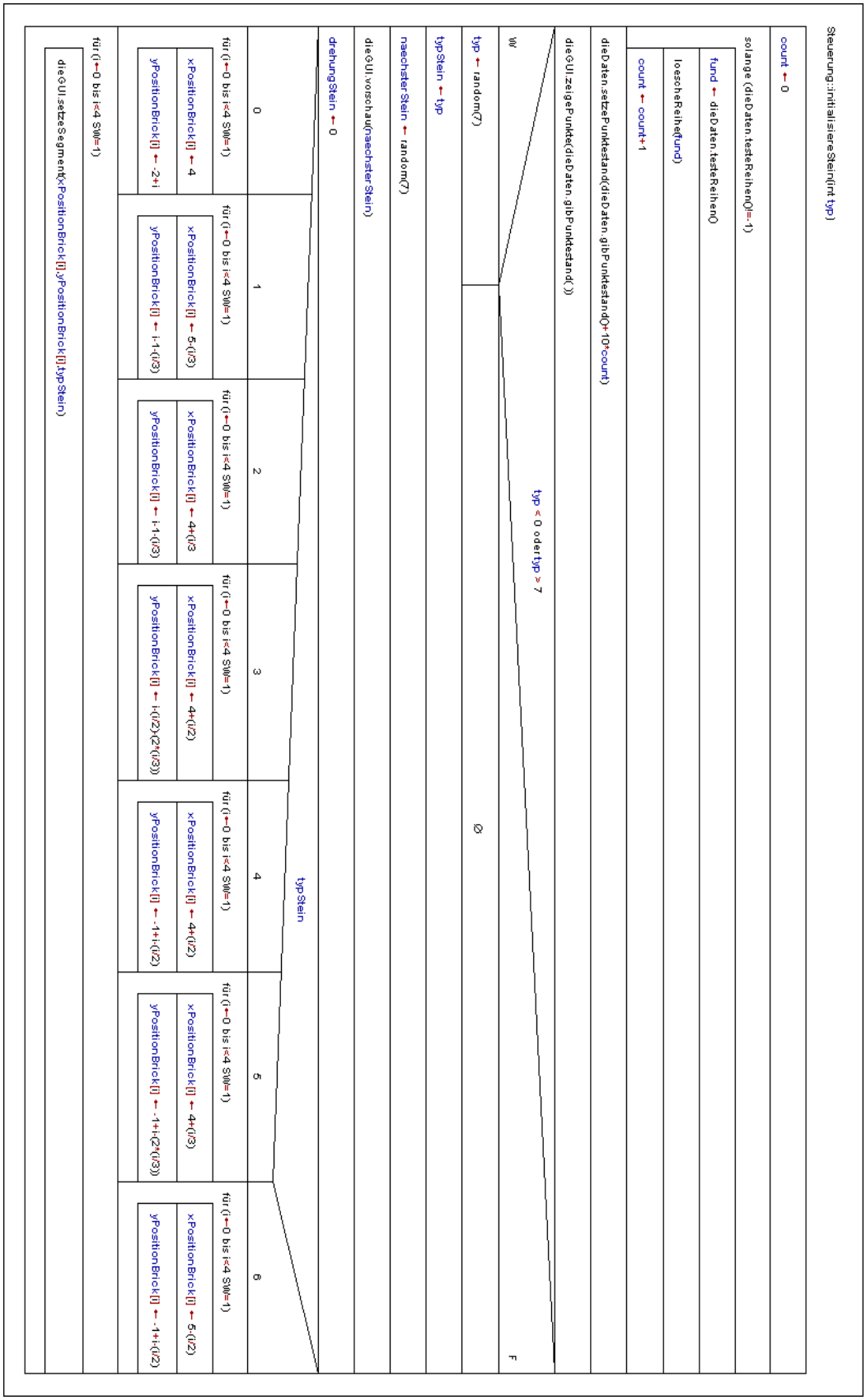


Teil 2:


F
∅
typStein =
1
2
3
4
5
6
drehungStein =
0
1
2
3
drehungStein =
0
1
2
3
kollision = false
W
F
kollision ← testePosition(xPositionBrick[1]-1,yPositionBrick[1]-1)
∅
kollision = false
W
F
kollision←testePosition(xPositionBrick[1]+1,yPositionBrick[1])
∅
kollision = false
W
F
kollision←testePosition(xPositionBrick[1]-1,yPositionBrick[1])
∅
kollision = false
W
F
dieGUI.loescheSegment(xPositionBrick[0],yPositionBrick[0])
yPositionBrick[0]←yPositionBrick[1]-1
xPositionBrick[0] ← xPositionBrick[1]-1
dieGUI.setzeSegment(xPositionBrick[0],yPositionBrick[0],typStein)
dieGUI.loescheSegment(xPositionBrick[2],yPositionBrick[2])
yPositionBrick[2] ← yPositionBrick[1]
xPositionBrick[2] ← xPositionBrick[1]+1
dieGUI.setzeSegment(xPositionBrick[2],yPositionBrick[2],typStein)
dieGUI.loescheSegment(xPositionBrick[3],yPositionBrick[3])
yPositionBrick[3] ← yPositionBrick[1]
xPositionBrick[3] ← xPositionBrick[1]-1
dieGUI.setzeSegment(xPositionBrick[3],yPositionBrick[3],typStein)
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅

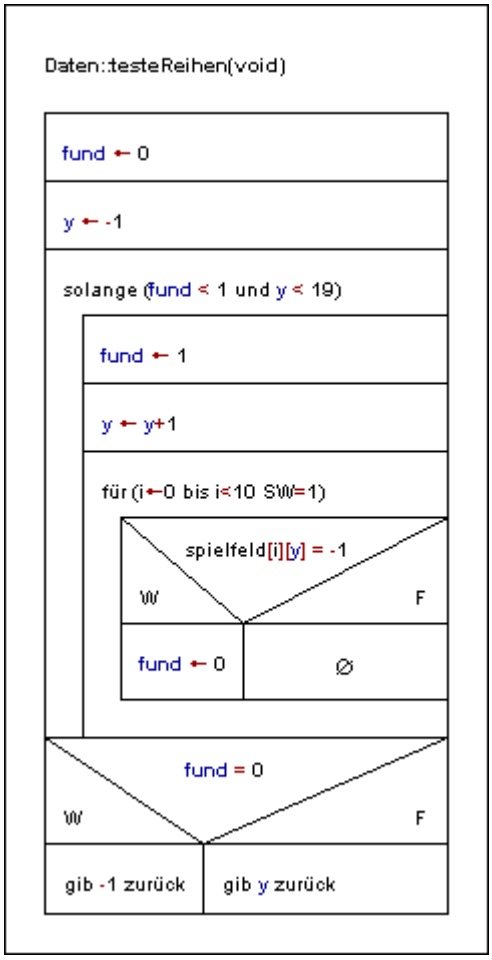
Operation initialisiereStein( ) der Steuerung

Eine bessere Ansicht finden Sie im Anhang.



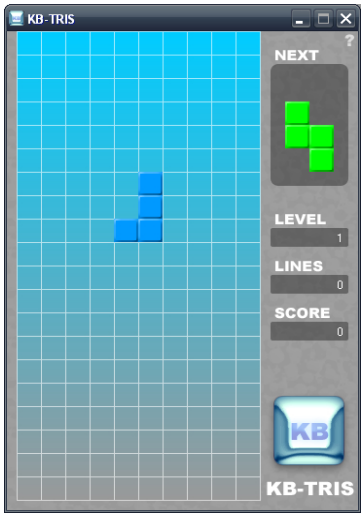
Operation testeReihen( ) der Daten

Ist der Rückgabewert -1, so gibt es keine vollen Reihen.
Die Variable Y gibt den y-Wert der vollen Reihe an.

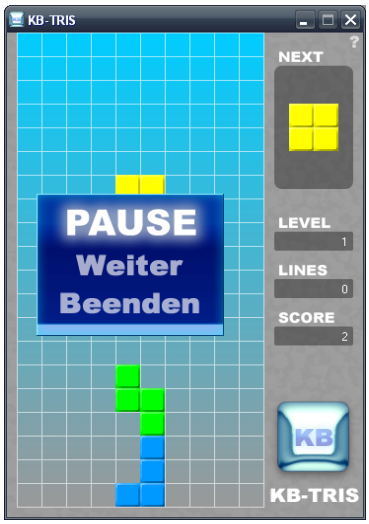


Zustände der GUI

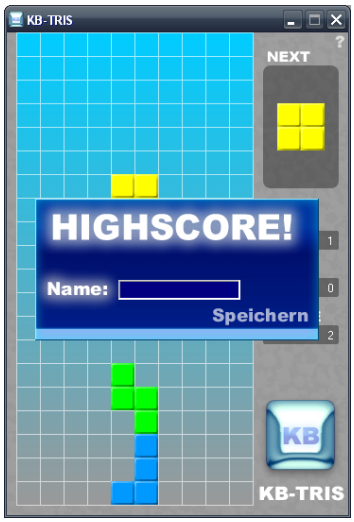
Das Menü



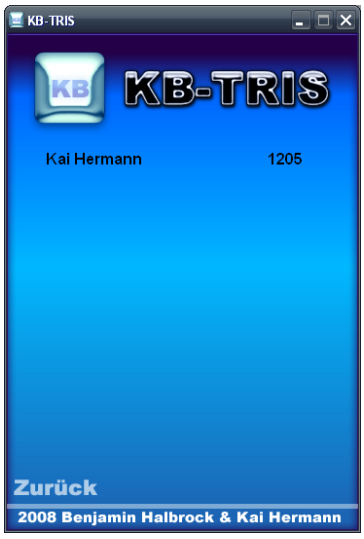
Im Spiel



Der Pausemodus



Eintragen des Highscores



Die Highscoreanzeige

Credits

Arbeitsverlauf

Zu Anfang des Projektes wurden im Lastenheft die Ziele formuliert.

Daraufhin setzten wir uns zusammen und entwickelten ein Klassendiagramm.

Nach diesem erstellte Benjamin einen Rahmencode, der als Basis für die weitere Programmierung diente.

Der Code wurde von Kai und Benjamin nach und nach entwickelt und eingepflegt, wodurch sich auch die Klassen, Operationen und Attribute veränderten.

Nach und nach entstand ein funktionstüchtiges Programm, welches den Anforderungen entsprach.

Kai überarbeitete dann die GUI und pflegte eine Highscoreverwaltung ein, während Benjamin sich um die Dokumentation kümmerte.

Außer der Beschreibung der GUI auf Seite 7 wurde die Dokumentation von Benjamin erstellt.

Arbeitsverteilung:

Lastenheft:

Benjamin 45 Minuten

Entwurfsprozess:

Kai: 3 Stunden
Benjamin: 3 Stunden

Code Entwicklung:

Kai: 9 Stunden
Benjamin: 5 Stunden

Fehlersuche:

Kai: 3 Stunden

Diagramme:

Kai: 20 Minuten
Benjamin: 5 Stunden

Dokumentation:

Kai: 20 Minuten
Benjamin: 2 Stunden

Verwendete Programme

Borland C++ Builder 6:

- Erstellung der GUI
- Programmierung
- http://www.codegear.com

Adobe Fireworks CS3 Demo:

- Erstellung der Grafiken
- http://www.adobe.com/go/tryfireworks_de

OpenOffice:

- Erstellung der Dokumentation
- http://www.openoffice.org

StarUML:

- Erstellung des Klassendiagramms
- Erzeugung des Coderahmens
- Erstellung der Sequenzdiagramme
- http://staruml.sourceforge.net/

Structorizer:

- Erstellung der Struktogramme
- http://structorizer.fish.lu

Zusammenfassung

Verlangt war eine Umsetzung des Spielprinzips von Tetris mit 7 verschiedenen Steinen.

Diese sollten fallen und sich drehen und verschieben lassen.

Des weiteren sollte sie sich stapeln und eine volle Reihe sollte gelöscht werden.

Auch eine Punkteanzeige und ein Game-Over-Dialog waren umzusetzen.

Dies alles ist uns gelungen.

Sogar die optionale Highscore Anzeige wurde eingebaut.

Die Umsetzung des Spieles erwies sich an manchen stellen komplexer als erwartet und wir hatten mit manchem Fehler zu kämpfen.

Auch kosteten manche Dinge mehr Zeit als erwartet.

Wenn wir mehr Zeit zur Verfügung gehabt hätten, hätten wir bestimmt noch einige Spielereien eingebaut.

So hatten wir die Idee, dass ein Packman ein „Tetris“(4Reihen auf einmal) frisst.

Und auch eine von der Fallgeschwindigkeit abhängige Hintergrundmusik zählt zu unseren nicht umgesetzten Ideen.

Abschließend betrachtet, war Tetris genau das richtige Spiel für uns.

Wir hatten etwas zum Denken, Knobeln und Ausprobieren.

Anhang

Dateien:

Dokumente:

- Diese Dokumentation
- Das Lastenheft

Das Klassendiagramm

Das Sequenzdiagramm zu neuesSpiel( )

Das Sequenzdiagramm zu fallen ( ) und das Spiel geht weiter

Struktogramme: (zum Bearbeiten der Dateien auch das Tool Structorizer)

- bool Steuerung_testePosition
- Steuerung_aktualisiereSpielfeld
- Steuerung_loescheReihe
- Steuerung_schreibeDaten
- Daten_testeReihen
- Steuerung_drehen
- Steuerung_drehen_kurz
- Steuerung_fallen
- Steuerung_initialisiereStein
- Steuerung_tasteGedruecktRechts

Eingabebuffer

http://www.c-plusplus.de/forum/topic,9707.html

Wir haben uns dazu entschlossen die Pfeiltasten mit folgendem Code abzufragen, da dies nach ausführlicher Recherche die Einzige Möglichkeit zu sein schien dies zu tun.

VCL_MESSAGE_HANDLER(CM_DIALOGKEY, TWMKey, CMDialogKey)

END_MESSAGE_MAP(TdieGUI)

void __fastcall CMDialogKey(TWMKey &AMsg);

...

Erstellung eines eigenen Designs

Um ein eigenes Design zu erstellen sollte zuerst eine Kopie des „system\gfx-Ordners“ angelegt werden.

Sie finden ihn im Unterordner ihrer KB-TRIS Installation.

Danach können sie die Grafiken mit einem beliebigen Bildbearbeitungsprogramm bearbeiten,

Schon haben sie ein eigenes Design erstellt.

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Abschnitte, die wörtlich oder annähernd wörtlich aus einer Veröffentlichung entnommen sind, als solche kenntlich gemacht habe, ferner, dass die Arbeit noch nicht veröffentlicht und auch keiner anderen Prüfungsbehörde vorgelegt worden ist.

Pforzheim, den _____

_____

Vorname Nachname