Sokoban 3D

Dokumentation der Hausarbeit
Computergrafik II

Fachhochschule Bielefeld
Fachbereich Ingenieurwissenschaften und Mathematik

3. März 2013

Aleksandr Epp	112 320	aleksandr.epp@fh-bielefeld.de
Wjatscheslaw Sawtschenko	112 603	wjatscheslaw.sawtschenko@fh-bielefeld.de

Dozent:	Prof. Dr. Wolfgang Bunse
Veranstaltung:	Computergrafik
Semester:	SS 2012

Inhaltsverzeichnis

I Einleitung **4**

1 Was ist Sokoban? **4**

2 Spielprinzip **4**

3 Entwicklungsumgebung **4**

II Bedienung **5**

4 Kamera **6**

5 Bewegung auf dem Level **6**

6 Ende des Levels **7**

III Implementierung **8**

7 Grundgerüst **8**

8 Level **8**

9 Spielobjekte **9**

9.1 Leere Felder 9

9.2 Speilfelder 9

9.3 Wände 10

9.4 Zielfelder 10

9.5 Kisten 10

9.6 Spieler 11

10 Erweiterbarkeit **11**

10.1 Spielumfang 11

10.2 Spielobjekte 11

10.3 Spielregeln 11

IV Funktionen **12**

11 Level initialisieren **12**

12 Textausgabe 12

13 Texturen laden mit SOIL 13

14 Skinning, Hilfe und Legende 13

V Anhang 14

Teil I

Einleitung

Im Rahmen der Hausarbeit wird ein "Sokoban"-Spiel mit OpenGL entwickelt.

In erster Linie soll das Spielprinzip realisiert werden. Aus diesem Grund werden keine komplexe Modelle verwendet und das ganze Spiel ist mit Würfeln aufgebaut.

Als Grundlage für die Anwndung dienen Beispiele aus der OpenGL Super Bible. Das gesammte Projekt wird mit GLUT entwickelt. Auch für die Textausgaben wurden keine Windows-Funktionen benutzt, damit das Projekt plattformunabhängig bleibt.

1 Was ist Sokoban?

Sokoban (japanisch „Lagerhausverwalter") ist ein Computerspiel, das von Hiroyuki Imabayashi entwickelt und 1982 erstmals für verschiedene Computersysteme veröffentlicht wurde.

2 Spielprinzip

In einem einfachen Spielprinzip gilt es, mit einer Spielfigur alle Kisten nacheinander auf die dafür vorgesehenen Zielfelder zu bewegen, wobei es keine Vorgabe gibt, welches Objekt auf welches Zielfeld bewegt werden soll. Die Kisten können von der Spielfigur nur geschoben und nicht gezogen werden, ein Verschieben mehrerer Kisten zugleich ist nicht möglich. Die einzig möglichen Bewegungsrichtungen der Spielfigur sind nach vorne, hinten, rechts und links, so dass keine diagonalen Züge möglich sind.

3 Entwicklungsumgebung

Das Projekt wird in **DEV-C++** mit **MinGW**-Bibliotheken unter Windows 7 entwickelt.

Teil II

Bedienung

Nach dem der Benutzer das Spiel gestartet hat, gelangt man zum ersten Level:

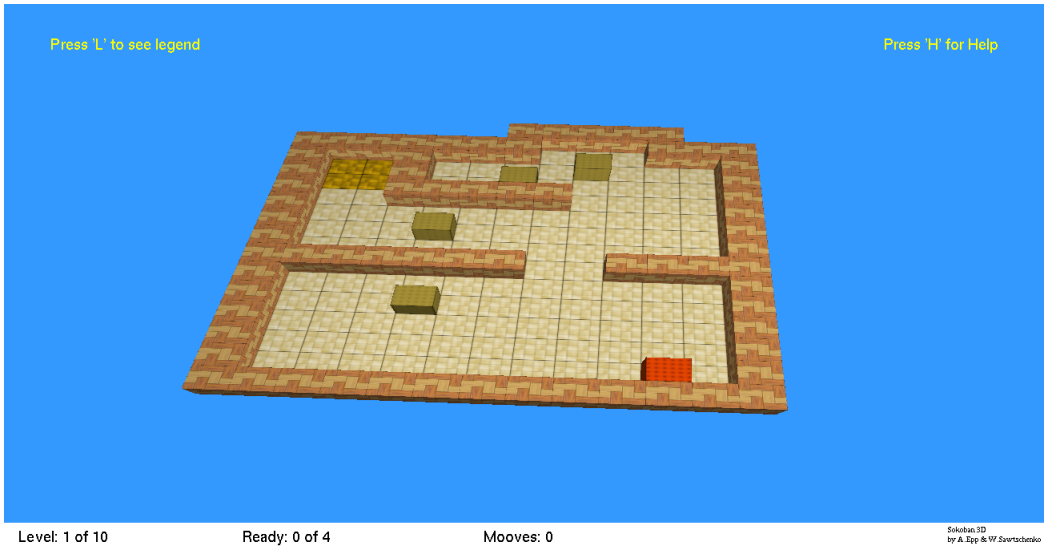


Abbildung 1: Spielstart

Das Spiel-Fenster kann in vier Bereiche unterteilt werden: *Legende*, *Hilfe*, *Info-Leiste* und das *Spielfeld* selbst. Beim ersten Start des Spiels wird der Benutzer evtl. noch nicht wissen, was er zu tun hat und kann sich die *Legende* anschauen oder die *Hilfe* aufrufen.

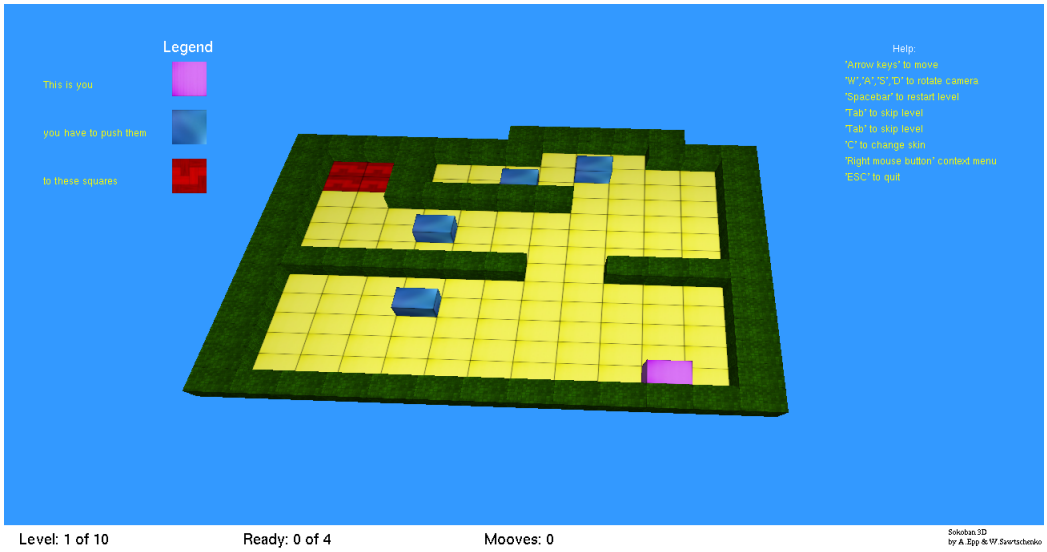


Abbildung 2: Hilfe & Legende

In der *Info-Leiste* werden Daten für das aktuelle Level ausgegeben: die Level-Nummer, die Anzahl der Kisten, die sich schon im Zielfeld befinden, und die Anzahl der auf diesem Level gemachten Schritte. Mit der rechten Maustaste öffnet man ein kleines Menu mit drei Optionen: Level neustarten, Kamrea zurücksetzen und Spielende. In dieser

Version des Spiels besteht die Möglichkeit beliebig viele Level zu überspringen oder zu schon abgeschlossenen Level zurückzukehren. Außerdem wurde eine **Skin**-Funktion implementiert, welche es erlaubt **Texture-Packs** zu wechseln.

'H'	Schaltet die Hilfe ein und aus.
'L'	Schaltet die Legende ein und aus.
'C'	Wechselt den "Skin"
'ESC'	Beendet das Spiel.
Rechte **Maustaste**	Öffnet ein kleines Spielmenü.
'Tab'	Springt zum nächsten Level. (nicht beim letzten)
'Backspace'	Geht ein Level zurück. (nicht beim ersten)
'Leertaste'	Level neustarten.

Tabelle 1: Allgemeine Steuerung

4 Kamera

Die Möglichkeit die Kamera zu bewegen wurde aus dem Beispielprogramm der Super Bible **ShadowMap.c** übernommen und ermöglicht es dem Benutzer das Spielfeld aus verschiedenen Blickwinkel zu betrachten.

'W'	Bewegt die Kamera nach oben.
'A'	Bewegt die Kamera nach links.
'S'	Bewegt die Kamera nach unten.
'D'	Bewegt die Kamera nach rechts.

Tabelle 2: Kamerasteuerung

5 Bewegung auf dem Level

Wie schon erwähnt, kann der Spieler sich in die Richtungen nach vorne, hinten, rechts und links bewegen. Sobald man eine Kiste berührt kann diese geschoben werden.

$\uparrow$	Bewegt den Spieler nach oben.
$\leftarrow$	Bewegt den Spieler nach links.
$\downarrow$	Bewegt den Spieler nach unten.
$\rightarrow$	Bewegt den Spieler nach rechts.

Tabelle 3: Bewegungssteuerung

6 Ende des Levels

Ein Level ist zu Ende, wenn alle Kisten in den Zielfelder stehen. Wenn die Kiste das Zielfeld erreicht, ändert sie ihre Farbe und die "Ready"-Zahl in der *Info-Leiste* erhöht sich. Nachdem der Spieler alle Figuren zum Ziel gebracht hat, erscheint eine Meldung und mit der "Enter"-Taste gelangt man zum nächsten Level. Die Abb. 2 zeigt diese Spielsituation.

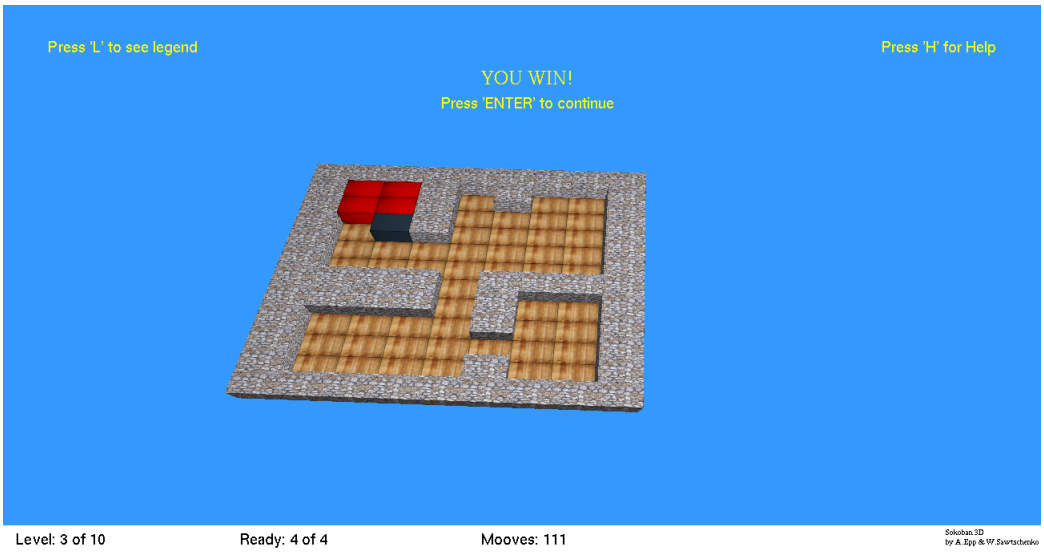


Abbildung 3: Level fertig

Teil III

Implementierung

7 Grundgerüst

Das Spielfeld besteht insgesamt aus 400 Felder. Die XY-Koordinaten für jedes Feld sind in einem zweidimensionalem Array gespeichert. Dieses Array findet man in der Datei **field.c** unter dem Namen **field**. Durch das Speichern der Koordinaten in solch einem Array, kann man jedes Feld durch seine Nummer ansprechen, ohne die Koordinaten ausrechnen zu müssen.

Ein Feld ist ein Quadrat mit der Seitenlänge 30 Einheiten und die gespeicherten Koordinaten eines Feldes geben die Mitte dieses Quadrats an. Die Nummerierung der Felder fängt links oben mit 0 (X: -285 , Y: -285) an und endet unten rechts mit 399 (X: 285 , Y: 285). Wenn der Spieler sich nach links oder rechts bewegt, wird seine aktuelle Position um 1 erhöht/erniedrigt. Wenn die Bewegung nach vorne oder nach hinten ausgeführt wird, wird seine Position um eine ganze Zeile erhöht/erniedrigt, d.h. um 20.

Auf diesen Felder, können die Spielobjekte aus Tab. 4 gezeichnet werden. Es wurde eine Notation verwendet, welches es ermöglicht die Level in einem Texteditor zu erstellen.

8 Level

Die Level mit den Startpositionen von Kisten und dem Spieler werden in der Datei **field.c** in **initLevel** als String hitereinander gespeichert. Durch **\N** wird ende des Levels gekennzeichnet. Für die Level-Erstellung wurde Excell verwendet. Das gezeichnete Level lässt sich ohne weitere Formatierung einfach in die **field.c** einfügen. Das **initLevel** wird am Anfang des Levels, bei einem Neustart des Levels und bei **Tab**/**Backspace** ausgelesen und alle Spielobjekte werden ermittelt und in die entsprechende Arrays gespeichert. In diesen Arrays stehen die Felder auf den sich die Objekte gerade befinden.

		-285	-255	-225	-195	-165	-135	-105	-75	-45	-15	15	45	75	105	135	165	195	225	255	285			
29																								
30		-285	-255	-225	-195	-165	-135	-105	-75	-45	-15	15	45	75	105	135	165	195	225	255	285		Level	1
31	-285	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	\		
32	-255	G	G	G	G	G	G	G	G	G	1	1	1	1	1	G	G	G	G	G	G	\		
33	-225	G	G	G	1	1	1	1	1	1	1	0	0	0	1	1	1	G	G	G	G	\		
34	-195	G	G	G	1	2	2	1	0	0	0	0	3	0	0	0	1	G	G	G	G	\		
35	-165	G	G	G	1	2	2	1	0	0	3	0	0	0	0	0	1	G	G	G	G	\		
36	-135	G	G	G	1	0	0	1	1	1	1	1	0	0	0	0	1	G	G	G	G	\		
37	-105	G	G	G	1	0	0	0	0	0	0	0	0	0	0	4	1	G	G	G	G	\		
38	-75	G	G	G	1	0	0	0	3	0	0	0	0	0	0	0	1	G	G	G	G	\		
39	-45	G	G	G	1	0	0	0	0	0	0	0	0	0	0	0	1	G	G	G	G	\		
40	-15	G	G	G	1	1	1	1	1	1	1	0	0	1	1	1	1	G	G	G	G	\		
41	15	G	G	G	1	0	0	0	0	0	0	0	0	0	0	0	1	G	G	G	G	\		
42	45	G	G	G	1	0	0	0	3	0	0	0	0	0	0	0	1	G	G	G	G	\		
43	75	G	G	G	1	0	0	0	0	0	0	0	0	0	0	0	1	G	G	G	G	\		
44	105	G	G	G	1	0	0	0	0	0	0	0	0	0	0	0	1	G	G	G	G	\		
45	135	G	G	G	1	0	0	0	0	0	0	0	0	0	4	0	1	G	G	G	G	\		
46	165	G	G	G	1	1	1	1	1	1	1	1	1	1	1	1	1	G	G	G	G	\		
47	195	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	\		
48	225	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	\		
49	255	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	\		
50	285	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	N\		

Abbildung 4: Erstes Level

9 Spielobjekte

Es wurde folgende Notation verwendet:

Objekt	Notation
Leeres Feld	G
Spielfeld	0
Wände	1
Zielfelder	2
Kisten	3
Spieler	4

Tabelle 4: Spielobjekte

9.1 Leere Felder

In der ersten version des Spiels, gab es keine Felder, die nicht verwendet werden, und somit konnte das Spielfeld nur eine Form haben. Es wurden also Koordinaten für noch mehr Felder ausgerechnet, und **G**-Felder eingefügt. Somit hat man die Möglichkeit die Form des Levels frei zu gestalten.

9.2 Speilfelder

Die Spielfelder werden in erster Linie nur dafür benutzt, die GL_QUADS mit der Bodentextur an der richtigen Stelle zu zeichnen. Im Laufe der Entwicklung wurde aber die Funktion eingefügt, dass die Kisten ihre Farbe wechseln, wenn sie auf dem Zielfeld stehen bzw. ihre Farbe zurückwechseln, wenn sie wieder das Zielfeld verlassen haben. Die Spielfelder werden also darauf überprüft, ob sich eine Kiste darauf befindet oder nicht.

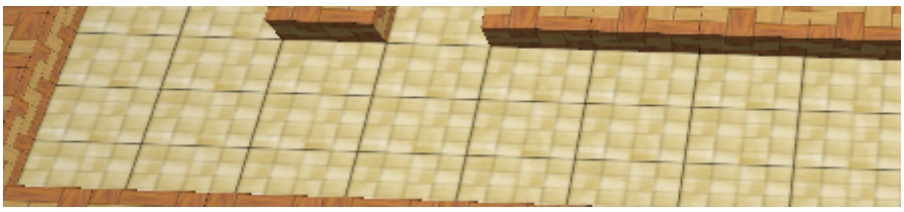


Abbildung 5: Speilfelder

9.3 Wände

Die Wände machen aus einem unendlich großen Raum ein richtiges Spielfeld, welches der Spieler nicht verlassen kann, solange eine Wand im Weg ist.

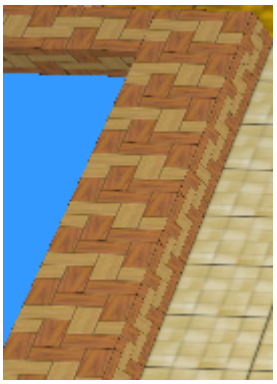


Abbildung 6: Wände

9.4 Zielfelder

Die Zielfelder werden bei jedem **RenderScene** darauf geprüft, ob und wieviele Kisten sich darauf befinden, und geben dem Benutzer ggf. Bescheid, dass das Level gewonnen wurde.

Abbildung 7: Zielfelder

9.5 Kisten

Anzahl der Kisten muss gleich der Anzahl der Zielfelder sein, sonst kann das Level nicht abgeschlossen sein. Die Kisten ändern ihre Farbe (die Texturen werden mit der roten Farbe vermischt)

wenn sie das Zielfeld erreicht haben.

(a) Kiste: auf dem Spielfeld

(b) Kiste: auf dem Zielfeld

Abbildung 8: Kisten

9.6 Spieler

Bewegung des Spielers und Interaktion zwischen den Spielobjekten und dem Spieler funktioneirt wie folgt:

- Wenn der Benutzer eine Bewegungstaste drückt, wird die Bewegung in jedem Fall ausgeführt, und der Spieler-Würfel an der neuen Position gezeichnet.
- Anschließend wird auf Kollisionen geprüft:
  - Wenn der Spieler sich jetzt "in" einer Wand befindet, wird er auf die vorherige Position gesetzt.
  - Wenn der Spieler sich jetzt "in" einer Kiste befindet, wird die Kiste in die gleiche Richtung bewegt.
    * Wenn die Kiste sich aber jetzt "in" einer anderen Kiste oder Wand befindet, werden beide, Spieler und die bewegte Kiste, zurückgesetzt.

Abbildung 9: Spieler

10 Erweiterbarkeit

10.1 Spielumfang

Das Spielfed, das momentan aus 400 Felder besteht kann beliebig vergrößert werden. Dazu müssten die Koordinaten für die neuen Felder ausgerechnet werden und die Bewegung des Spielers nach vorne/hinten angepasst werden, es sei denn die Zeilenlänge würde gleich bleiben (20).

Das Spiel hat 10 Level. Dies kann leicht geändert werden, indem neue Level in **field.c** in den **initLevel**-String eingngefügt werden. Beispiel eines Levels kann der Abb. 4 entommen werden.

10.2 Spielobjekte

Da die Koordinaten aller Spielobjete bekannt sind, können die Würfel durch andere Objekte ersetzt werden (z.B. Tannenbäume (Zylinder+Kegel) statt Wände und ein Schneeman (drei Kugeln) als Spieler).

Außerdem, können weitere Spielobjekte eingeführt werden, wie z.B. Teleporter oder Türen, die nur in eine Richtung aufgehen.

10.3 Spielregeln

Es wäre auch denkbar die Zeit einzuführen um z.B. Highscore zu ermitteln oder den Spieler aufzufordern ein Level in einer bestimmten Zeit zu beenden um zu gewinnen.

Teil IV

Funktionen

11 Level initialisieren

In der Variable **levelNum** wird die aktuelle Levelnummer gespeichert. Wenn ein Level gewonnen wurde, wird die variable inkrementiert und aus **initLevel** wird nächstes Level ausgelesen. Dann werden Positionen aller Spielobjekte sowie deren Anzahl in **calcCount()** initialisiert. Pro Spielobjektgruppe gibt es ein Array mit Position aller Objekte von diesem Typ (Nummer des Feldes) und eine **count**-Variable, die die Anzahl der Objekte von diesem Typ speichert.

12 Textausgabe

Textausgabe wurde mit standard GLUT-techniken realisiert, ohne Windows-API:

- **glutBitmapCharacter(font, *c);**

Zwei Hilfsfunktionen machen Text "statisch" (= Texte bewegen sich nicht mit der Kamera, sondern sind immer auf der gleichen Stelle):

- **glEnter2D(void)**
- **glLeave2D(void)**

Zunächst wird das "2D-Modus" gestartet. GLUT stellt 7 Schriften zur verfügung:

```
LPVOID glutFonts(7) = {
  GLUT_BITMAP_9_BY_15,
  GLUT_BITMAP_8_BY_13,
  GLUT_BITMAP_TIMES_ROMAN_10,
  GLUT_BITMAP_TIMES_ROMAN_24,
  GLUT_BITMAP_HELVETICA_10,
  GLUT_BITMAP_HELVETICA_12,
  GLUT_BITMAP_HELVETICA_18
};
```

Mit einem **sprintf** wird Text in ein Buffer geschrieben und dieser Buffer wird dann an die **renderBitmapString** übergeben. Die **renderBitmapString(float x, float y, void *font, char *string)** bekommt noch die Koordinaten des zu zeichnenden textes und die Schrift, und gibt anschließend den Text aus.

13 Texturen laden mit SOIL

Wegen der einfachen Anwendung werden alle Texturen mit **SOIL** geladen. **SOIL** ist eine freie Bibliothek, die das Laden der Bilder mit nur einer Zeile ermöglicht:

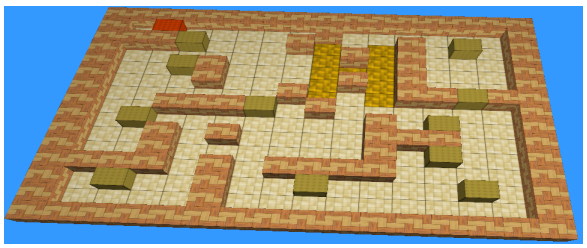
- GLuint tex = SOIL_load_OGL_texture("texture.png",0,1,16 | 2 | 1);
  - Als Parameter wird der Pfad zum Bild übergeben sowie eine Reihe von weiteren Parameter, die angeben in welchem Format das Bild vorliegt (RGB, RGBA usw.), ob Invertierung an der Y-Achse durchgeführt werden soll etc. Auflistung und genaue Beschreibung aller Flags ist in **soil.h**.

Allerdings funktioniert die von uns verwendete SOIL-Bibliothek nicht ganz korrekt, und vergibt jeder Textur die gleiche ID. D.h. die vorherige Textur wird immer überschrieben, auch wenn diese in einer anderen Variable gespeichert wurde. Aus diesem Grund müssen alle Texturen beim Rendern geladen werden. Dies ist zwar keine gute Lösung, doch bei unserem kleinem Spiel hat es gar keine Auswirkung auf die Performance.

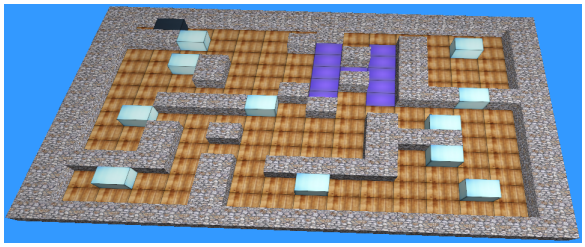
14 Skinning, Hilfe und Legende

Tastatur-Events werden abgefangen und je nachdem welche Taste gedrückt wurde wird die ensprechende Aktion ausgeführt:

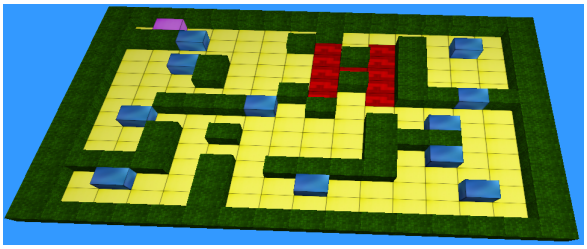
1. Je nach aktuellem Zustand, wird die Hilfe angezeigt oder versteckt.
2. Je nach aktuellem Zustand, wird die Legende angezeigt oder versteckt.
3. Je nach aktuellem Zustand, werden nächste Texturen geladen. Die Texturen-Pfade sind in Arrays hinterlegt. Zur zeit gibt es drei "Packs":



(a) Skin 1



(b) Skin 2



(c) Skin 3

Abbildung 10: Skins

Teil V

Anhang

Abbildungsverzeichnis

1 Spielstart 5
2 Hilfe & Legende 5
3 Level fertig 7
4 Erstes Level 8
5 Speilfelder 9
6 Wände 10
7 Zielfelder 10
8 Kisten 10
9 Spieler 11
10 Skins 13

Tabellenverzeichnis

1 Allgemeine Steuerung 6
2 Kamerasteuerung 6
3 Bewegungssteuerung 6
4 Spielobjekte 9

Literatur

(1) GLUT Dokumentation

(2) SuperBible, Third Edition by Richard S. Wright Jr., Benjamin Lipchak

(3) Wikipedia (Sokoban Beschreibung)

(4) SOIL Homepage `http://www.lonesock.net/soil.html`