

Aufgabe 5: Die Paderbox

Lösungsidee

Eine knifflige Aufgabe! Und vor allem eine Aufgabe, die man mehrmals liest und immer noch nicht verstanden hat – doch die Zeit ist vergangen und wir haben uns besprochen. Wir hoffen jetzt einen richtigen Lösungsansatz und dann natürlich auch eine korrekte Lösung geschaffen zu haben:

Für alle Zufälle legen wir fest, dass die Fälle als gleich wahrscheinlich angenommen werden können (zum Beispiel, dass am Anfang eine LED aus oder an ist; wir können diese Vorgänge also als „Laplace-Experimente“ ansehen).

Zufällige AN/AUS-Schaltung der Lampen vornehmen.

Für jede Lampe besteht dabei die Möglichkeit entweder an- oder ausgeschaltet zu sein.

Für die AN/AUS-Schaltung gibt es also $2^3 = 8$ verschiedene Möglichkeiten.

AN/AUS-Schaltung der Lampen speichern.

Zufälligen Anfangszustand aus den drei Zuständen auswählen ohne eine Änderung der Lampen.

Nummer des Anfangszustandes speichern.

Programmieren der Tasten:

Für jede der beiden Tasten muss einem alten Zustand ein neuer Zustand zugewiesen werden.

Beispiel: (1;2), (2;1), (3;3)

Ein Paar (2-Tupel) beinhaltet bei dieser Darstellung jeweils die Nummer des alten Zustandes und die Nummer des neuen Zustandes.

Zuweisung speichern.

Die Frage, die wir uns gestellt haben war, ob denn ein neuer Zustand auch der alte Zustand wieder sein kann. Des Weiteren tauchte die Frage auf, ob ein ein neuer Zustand mehrmals auftauchen darf – wie z.B. bei (1;2), (2;2) – oder nicht. Wir entschieden uns diese Fragen jeweils mit ja zu beantworten.

Festlegen der Beziehung Lampen \leftrightarrow Zustände:

1 Lampe mit 2 Zuständen verbunden; 1 Zustand mit 2 Lampen verbunden

Lampen: L_1, L_2, L_3 ; Zustände: Z_1, Z_2, Z_3

Beziehung wird anhand der Lampen-“Zugehörigkeit“ zu den Zuständen festgelegt.

Z_1 : zufällig 2 Lampen auswählen (L_a, L_b)

Z_2 : zufällig eine Lampe von Z_1 (L_a oder L_b) auswählen + L_c

Z_3 : übrige Lampe, die noch nicht mit Z_1 und Z_2 verbunden ist, und L_c auswählen

am Beispiel veranschaulicht:

	Z_1	Z_2	Z_3
L_1	■	□	■
L_2	□	■	■
L_3	■	■	□

Gespeichert werden muss zudem

- der aktuelle Zustand und
- die aktuelle AN/AUS-Schaltung der Lampen.

Alternativ zur aktuellen AN/AUS-Schaltung der Lampen könnte auch die Abfolge der bisherigen Zustände gespeichert werden und die aktuelle AN/AUS-Schaltung der Lampen aus dieser Abfolge und der anfänglichen AN/AUS-Schaltung der Lampen berechnet werden.

Dabei stießen wir allerdings auf Fragen des Speicherplatzkriterium und taten diese Überlegung als unnötig ab – dadurch würde gegebenenfalls unerwünscht Redundanz im System auftreten.

Beim Rücksetzen in den Anfangszustand den Anfangszustand aus dem Speicher holen und dem aktuellen Zustand diesen Wert zuweisen.

Die Frage, die hier auftauchte war: AN/AUS-Schaltung der Lampen auch auf Anfang zurücksetzen? Dies würde ggf. nochmal separates Speichern benötigen. Wir entschieden uns allerdings dann doch dafür. (Dies war auch ein Punkt in den FAQ: „Bezieht sich das Rücksetzen nur auf den 'inneren' Zustand? - Die Aufgabenstellung überlässt diese Entscheidung den Bearbeiterinnen und Bearbeitern. Für das PB-Spiel ist es aber vielleicht netter, wenn nach einem 'Rücksetzen' alles ganz genau so ist wie am Anfang, damit man beim Raten wirklich noch mal ganz von vorn beginnen kann.“)

Beim Verraten der geheimen Programmierung:

- die Beziehungen zwischen den Zuständen und den Lampen,
- den Anfangszustand,
- und die Programmierung der Tasten ausgeben.

Bei der Umsetzung hatten wir schon Überlegung mit einem Mikrocontroller zu „arbeiten“. Jedoch sprach so einiges dagegen: ggf. müssten Platinen gelayoutet und geätzt werden. Zudem ist neben dem Programmieren und Compilieren noch eine Übertragung auf den Controller nötig. Wohl hätte dieses Projekt auch noch ausführlicher kommentiert werden müssen und das Einsenden von Hardware wäre wohl ein wirklicher Sonderfall – die Idee hätte uns aber gefallen. Für die Umsetzung blieb uns allerdings nicht wirklich Zeit, aber wer weiß, vielleicht bastelt sich der ein oder andere in nächster Zeit seine eigene Paderbox und kann damit so manchen verwirren!

Die Paderbox wurde im Großen und Ganzen von uns als Black Box gesehen: keiner weiß was drin ist und was sich innen abspielt (gut, als Programmierer jetzt schon und jemand der die Aufgabenstellung kennt), man sieht nur die LEDs, zwei Taster und den Reset-Knopf. Das Ganze ist also wunderbar vergleichbar mit einer Wahlmaschine!

Programm-Dokumentation

Entwicklungsumgebung: Dev-C++ 4.9.9.2

Verwendete Librarys:

- C/C++ Standardlibrarys
- SDL für die Grafik
- sdobject / MathFunc 2006 by Alexander Weld, eine Art SDL-API

Erklärung der einzelnen Klassen:

Lamp

Beschreibt eine Lampe.

Konstruktor

Die Lampe wird auf einen zufälligen Zustand gesetzt:

```
_stat = ( rand( ) % 2) ? true : false;
```

rand() % 2 ist entweder 0 oder 1, daher entweder ein oder aus.
_fstat speichert den Ausgangszustand.

Switch

schaltet die Lampe von ein auf aus und umgekehrt.

Reset

versetzt die Lampe in ihren Ausgangszustand

DrawMe

Zeichnet die Lampe, je nach Zustand gelb (=ein) oder grau (=aus).

State

Beschreibt einen Zustand.

Variablen

L₁, L₂ sind die zwei Lampen mit denen der Zustand verbunden ist.

Action

Schaltet die mit dem Zustand verbundenen Lampen um

DrawLines

Zeichnet zu den verbundenen Lampen Linien.

DrawMe

Zeichnet den Zustand. (Rot wenn er gerade an ist, weiß wenn nicht.)

PaderBox

Ist das Hauptelement, hier spielt sich die ganze Programmlogik ab.

Konstruktor

Es werden je 3 leere Lampen und Zustände geschaffen und ihnen Koordinaten zugewiesen.

Dann werden die Zustände mit einem bestimmten Muster mit den Lampen verbunden:

Zustand 1:

```
z1[0][0] = (int)MathFunc::rndnr(3);
```

Wählt für den ersten Zustand eine zufällige Lampe (L_a) aus.

```
while( z1[0][1] == z1[0][0] )
```

```
    z1[0][1] = (int)MathFunc::rndnr(3);
```

Wählt für den ersten Zustand eine andere zufällige Lampe (L_b) aus die nicht L_a ist. (Eventuell eine unschöne Lösung, da es theoretisch zu einer Endlosschleife kommen könnte wenn $z1[0][1]$ immer gleich $z1[0][0]$ ist.)

Zustand 2:

```
z1[1][0] = ( rand( ) % 2 ) ? z1[0][0] : z1[0][1];
```

Wählt zufällig eine Lampe von $Z1$ aus.

```
while( z1[1][1] == z1[0][0] || z1[1][1] == z1[0][1] )
```

```
    z1[1][1] = (int)MathFunc::rndnr(3);
```

Wählt $L_c \neq L_a$ oder L_b aus. (Genauso unschön wie die Anweisung oben...)

Zustand 3:

```
z1[2][0] = z1[1][1];
```

Die erste Lampe = L_c

```
z1[2][1] = (int)abs( (z1[1][1]) - (z1[1][0]) )-1;
```

Die zweite Lampe ist die, die im Moment nur noch mit einem Zustand verbunden ist.

```
zustand = (int)MathFunc::rndnr(3);
```

Zufälliger Anfangszustand wird ausgewählt...

```
fz = zustand;
```

...und gespeichert.

```
S[0].SetLamps( &L[z1[0][0]], &L[z1[0][1]] );
```

```
S[1].SetLamps( &L[z1[1][0]], &L[z1[1][1]] );
```

```
S[2].SetLamps( &L[z1[2][0]], &L[z1[2][1]] );
```

Den Zuständen werden ihre Lampen zugeordnet.

```
keys[0][0] = (int)MathFunc::rndnr(3);
```

```
keys[0][1] = (int)MathFunc::rndnr(3);
```

```
keys[0][2] = (int)MathFunc::rndnr(3);
```

```
keys[1][0] = (int)MathFunc::rndnr(3);
```

```
keys[1][1] = (int)MathFunc::rndnr(3);
```

```
keys[1][2] = (int)MathFunc::rndnr(3);
```

Die Tasten werden zufällig programmiert.

Reset

Die PaderBox wird in ihren Ausgangszustand zurückgesetzt.

SwitchLamps

Die Lampen des gewählten Zustands werden umgeschaltet.

KeyPress

Der neue Zustand wird ausgewählt, je nach dem welche Taste gedrückt wurde und welcher Anfangszustand herrschte.

PrintMatrix, PrintLamps, PrintKeys

Nur noch zu Debug-Zwecken enthalten → Keine Bedeutung.

DrawPB

Malt die „geheime“ PaderBox, d.h. Man sieht nur die LEDs und die Schalter (angedeutet durch 2 Kreise am linken Rand)

DrawPBSolution

Malt die „offene“ PaderBox, d.h. Man sieht die LEDs, die Zustände, mit ihren Verbindungen und die Schalter/Zustände Verbindungen.

Das Hauptprogramm

Am Anfang werden alle Klassen etc. erstellt und das restliche Programm initialisiert.

```
while( mainloop )
```

Leitet die Hauptschleife ein.

Zu Beginn wird der Bildschirm gelöscht und die Lizenz und die Hilfe angeschrieben.

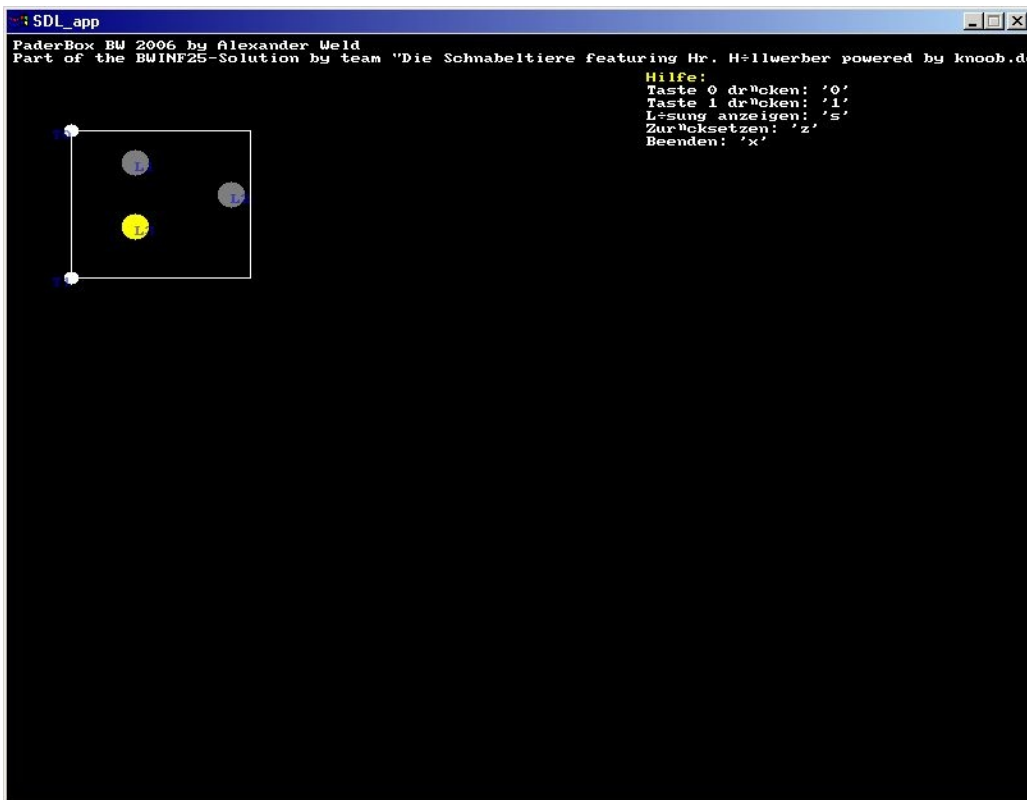
Dann wird abgefragt ob die Lösung angezeigt werden soll oder nicht. Wenn sie angezeigt werden soll wird noch eine extra Hilfe eingeblendet.

Danach kommt die Tastaturabfrage.

- Taste '0': In der PaderBox wird die Taste 0 gedrückt
- Taste '1': In der PaderBox wird die Taste 1 gedrückt
- Taste 'x': Das Programm wird beendet.
- Taste 's': Die Lösung wird angezeigt.
- Taste 'z' (Wegen amerikanischer Tastatur mit 'y' im Programm bezeichnet): Die PaderBox wird zurückgesetzt.

Am Ende wird alles de-initialisiert, eine Copyright Meldung ausgegeben, und beendet.

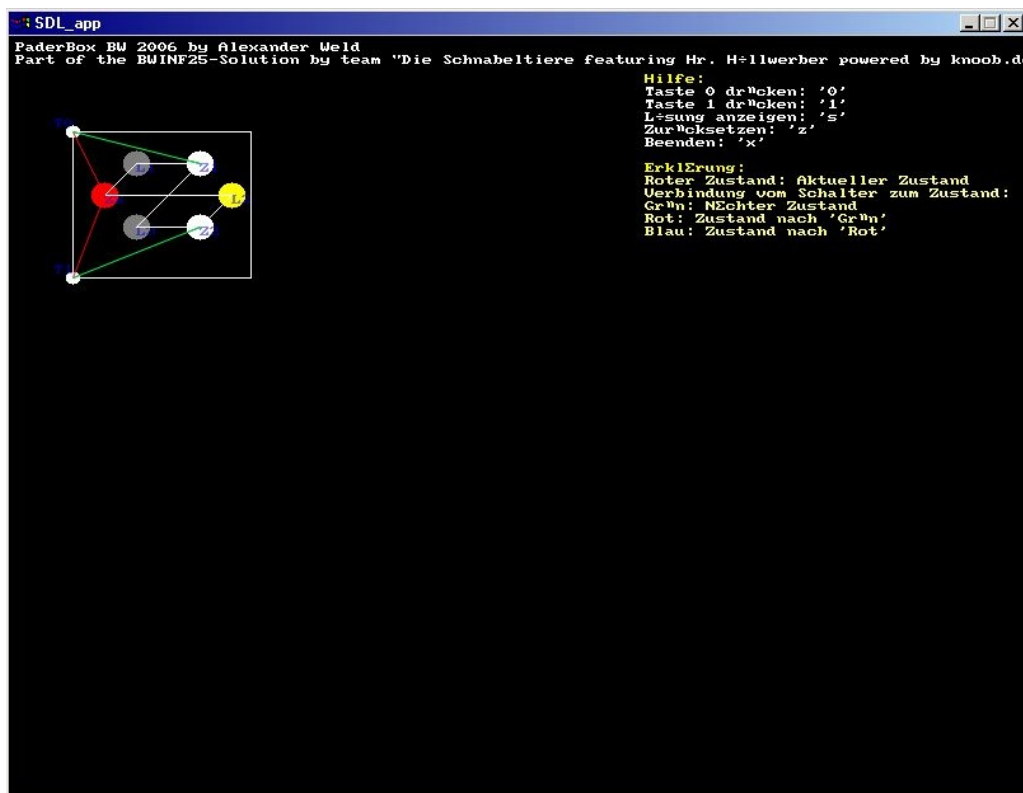
Programm-Ablaufprotokoll



Eine Lampe leuchtet



Die Lösung der Paderbox wurde eingeblendet



Eine Weitere Auflösung der Paderbox

Beispiel eines txt-logfiles:

```
PaderBox V0.2 BW 2006 by Alexander Weld
Hilfe per 'h'
Lampe: 1 : 1
Lampe: 2 : 0
Lampe: 3 : 1
Eingabe: h
```

Hilfe:

Es gilt die geheime Tastenprogrammierung herrauszufinden.
Dazu kann man folgende Sachen machen:

```
Taste 1: 1 - Taste 1 betötigen => Zustandswechsel
Taste 2: 2 - Taste 2 betötigen => Zustandswechsel
Zurücksetzen: z - In den Ausgangszustand zurückversetzen
Beenden: x - Programm Beenden
```

```
Lampe: 1 : 1
Lampe: 2 : 0
Lampe: 3 : 1
Eingabe: 1
Taste 1 gedrückt
Lampe: 1 : 1
Lampe: 2 : 1
Lampe: 3 : 0
Eingabe: 1
Taste 1 gedrückt
Lampe: 1 : 1
Lampe: 2 : 0
Lampe: 3 : 1
Eingabe: 1
Taste 1 gedrückt
Lampe: 1 : 1
Lampe: 2 : 1
Lampe: 3 : 0
Eingabe: 2
Taste 2 gedrückt
Lampe: 1 : 0
Lampe: 2 : 1
Lampe: 3 : 1
Eingabe: 2
Taste 2 gedrückt
Lampe: 1 : 1
Lampe: 2 : 1
Lampe: 3 : 0
Eingabe: 2
Taste 2 gedrückt
Lampe: 1 : 0
Lampe: 2 : 1
Lampe: 3 : 1
Eingabe: 2
Taste 2 gedrückt
Lampe: 1 : 1
Lampe: 2 : 1
Lampe: 3 : 0
Eingabe: 2
Taste 2 gedrückt
Lampe: 1 : 0
Lampe: 2 : 1
Lampe: 3 : 1
Eingabe: x
(1,1) = 2 (1,2) = 1
(2,1) = 2 (2,2) = 3
(3,1) = 3 (3,2) = 1
```

Taste 1

1 -> 3

2 -> 2

3 -> 2

Taste 2

1 -> 2

2 -> 3

3 -> 3

PaderBox BW 2006 by Alexander Weld

Part of the BWINF25-Solution by team "Die Schnabeltiere featuring Hr. H+llwerber
powered by knoob.de"

Drücken Sie eine beliebige Taste . . .

Programm

main.h

```
1  /*
2  PaderBox BW 2006 by Alexander Weld
3  Part of the BWINF25-Solution by team "Die Schnabeltiere featuring Hr. Höllwerber
   powered by knoob.de"
4
5  main.h
6
7  V0.0 11.11.2006
8  */
9  #include <cstdlib>
10 #include <iostream>
11 #include <string>
12 #include "MathFunc.h"
13 #include "sdlobject.h"
14 namespace PB
15 {
16 int b2i( bool X )
17 {
18     return (X) ? 1 : 0;
19 }
20 class Lamp
21 {
22     bool _stat;
23     bool _fstat;
24
25     SDL::coor *c;
26
27     SDL::color *c_off;
28     SDL::color *c_on;
29
30 public:
31
32     Lamp( void )
33     {
34         _stat = ( rand( ) % 2) ? true : false;
35         _fstat = _stat;
36         c = new SDL::coor( );
37         c_on = new SDL::color( 255, 255, 255, 127 );
38         c_off = new SDL::color( 255, 255, 0 );
39     }
40
41     ~Lamp( void )
42     {
43         delete c;
44         delete c_on;
45         delete c_off;
46     }
47
48     void SetCoor( double x, double y )
49     {
50         c->SetCoor( x, y );
51     }
52
53     SDL::coor *GetCoor( void )
54     {
55         return c;
56     }
57
58     bool Switch( void )
59     {
60         _stat = !_stat;
```

```
61         return _stat;
62     }
63
64     void Reset( void )
65     {
66         _stat = _fstat;
67     }
68
69     void SetStatus( bool X )
70     {
71         _stat = X;
72     }
73
74     bool GetStatus( void )
75     {
76         return _stat;
77     }
78
79     void DrawMe( SDL::SDL_Graphic *screen )
80     {
81         if( _stat )
82             screen->Circle( c_on, c, 10, true );
83         else
84             screen->Circle( c_off, c, 10, true );
85     }
86
87 };
88 class State
89 {
90     Lamp *L1,*L2;
91
92     SDL::coor *c;
93
94     public:
95
96     State( Lamp *l1, Lamp *l2 )
97     {
98         SetLamps( l1, l2 );
99         c = new SDL::coor( );
100     }
101
102     State( void )
103     {
104         c = new SDL::coor( );
105     }
106
107     ~State( void ) { }
108
109     void SetCoor( double x, double y )
110     {
111         c->SetCoor( x, y );
112     }
113
114     SDL::coor *GetCoor( void )
115     {
116         return c;
117     }
118
119     void SetLamps( Lamp *l1, Lamp *l2 )
120     {
121         L1 = l1;
122         L2 = l2;
123     }
124
125     Lamp *GetLamp( int i )
126     {
127
```

```
128         switch( i )
129         {
130             case 1:
131                 return L1;
132                 break;
133             case 2:
134                 return L2;
135                 break;
136             default:
137                 return NULL;
138                 break;
139         }
140     }
141
142     void Action( void )
143     {
144         L1->Switch( );
145         L2->Switch( );
146     }
147
148     void DrawLines( SDL::SDL_Graphic *screen )
149     {
150         SDL::color *temp;
151         temp = new SDL::color( );
152         screen->Line( temp, c, L1->GetCoor( ) );
153         screen->Line( temp, c, L2->GetCoor( ) );
154         delete temp;
155     }
156
157     void DrawMe( SDL::SDL_Graphic *screen, bool selected )
158     {
159         SDL::color *temp;
160         if( selected )
161             temp = new SDL::color( 255, 0, 0, 255 );
162         else
163             temp = new SDL::color( );
164         screen->Circle( temp, c, 10, true );
165         delete temp;
166     }
167 };
168 class PaderBox
169 {
170     Lamp *L;
171
172     State *S;
173
174
175     int z1[3][2]; //x => State, y => Lamp
176
177     int keys[2][3];
178
179     int zustand;
180     int fz;
181
182     public:
183
184     PaderBox( void )
185     {
186
187         L = new Lamp[3];
188         S = new State[3];
189
190         L[0].SetCoor( 100, 100 );
191         L[1].SetCoor( 175, 125 );
192         L[2].SetCoor( 100, 150 );
193         S[0].SetCoor( 150, 100 );
194         S[1].SetCoor( 150, 150 );
```

```
195         S[2].SetCoor( 75, 125 );
196
197
198
199
200         z1[0][0] = (int)MathFunc::rndnr(3);
201
202         while( z1[0][1] == z1[0][0] )
203             z1[0][1] = (int)MathFunc::rndnr(3);
204
205         z1[1][0] = ( rand( ) % 2 ) ? z1[0][0] : z1[0][0];
206
207         while( z1[1][1] == z1[0][0] || z1[1][1] == z1[0][1] )
208             z1[1][1] = (int)MathFunc::rndnr(3);
209
210         z1[2][0] = z1[1][1];
211         z1[2][1] = (int)abs( (z1[1][1]) - (z1[1][0]) )-1;
212
213         zustand = (int)MathFunc::rndnr(3);
214         fz = zustand;
215
216         S[0].SetLamps( &L[z1[0][0]], &L[z1[0][1]] );
217         S[1].SetLamps( &L[z1[1][0]], &L[z1[1][1]] );
218         S[2].SetLamps( &L[z1[2][0]], &L[z1[2][1]] );
219
220         keys[0][0] = (int)MathFunc::rndnr(3);
221         keys[0][1] = (int)MathFunc::rndnr(3);
222         keys[0][2] = (int)MathFunc::rndnr(3);
223         keys[1][0] = (int)MathFunc::rndnr(3);
224         keys[1][1] = (int)MathFunc::rndnr(3);
225         keys[1][2] = (int)MathFunc::rndnr(3);
226
227     }
228
229     ~PaderBox( void )
230     {
231         delete[] L;
232         delete[] S;
233     }
234
235     Lamp *GetLamp( int i )
236     {
237         return &L[i];
238     }
239
240     State *GetState( int i )
241     {
242         return &S[i];
243     }
244
245     int GetZustand( void )
246     {
247         return zustand+1;
248     }
249
250     void Reset( void )
251     {
252         zustand = fz;
253
254         L[0].Reset() ;
255         L[1].Reset() ;
256         L[2].Reset() ;
257
258     }
259
260     void SwitchLamps( void )
261     {
```

```
262         S[zustand].Action( );
263     }
264     void KeyPress( int n )
265     {
266
267         zustand = keys[n][zustand];
268         SwitchLamps( );
269     }
270     void PrintMatrix( void )
271     {
272         int i,j;
273         for( i = 0; i < 3; i++ )
274         {
275             for( j = 0; j < 2; j++ )
276                 std::cout << "(" << i+1 << "," << j+1 << ") = " << z1[i][j]+1 <<
" ";
277             std::cout << std::endl;
278         }
279     }
280
281     void PrintLamps( void )
282     {
283         int i;
284         for( i = 0; i < 3; i++ )
285             std::cout << "Lampe: " << i+1 << " : " << b2i( L[i].GetStatus() ) <<
std::endl;
286     }
287
288     void PrintKeys( void )
289     {
290         int i,j;
291         for( i = 0; i < 2; i++ )
292         {
293             std::cout << "Taste " << i+1 << std::endl;
294             for( j = 0; j < 3; j++ )
295                 std::cout << j+1 << " -> " << keys[i][j]+1 << std::endl;
296         }
297     }
298
299     void DrawPB( SDL::SDL_Graphic *screen )
300     {
301         SDL::coor Taste1( 50, 75 );
302         SDL::coor Taste2( 50, 190 );
303
304         SDL::color temp( 255, 255, 255, 255 );
305
306         L[0].DrawMe( screen );
307         screen->SimpleText( std::string( "L1" ), (int)L[0].GetCoor( )->GetX( ),
(int)L[0].GetCoor( )->GetY( ), 0, 0, 255, 127 );
308         L[1].DrawMe( screen );
309         screen->SimpleText( std::string( "L2" ), (int)L[1].GetCoor( )->GetX( )
, (int) L[1].GetCoor( )->GetY( ), 0, 0, 255, 127 );
310         L[2].DrawMe( screen );
311         screen->SimpleText( std::string( "L3" ), (int)L[2].GetCoor( )->GetX( ),
(int)L[2].GetCoor( )->GetY( ), 0, 0, 255, 127 );
312
313         screen->SimpleLine( 50, 75, 50, 190, 255, 255, 255, 255 );
314         screen->SimpleLine( 50, 190, 190, 190, 255, 255, 255, 255 );
315         screen->SimpleLine( 190, 190, 190, 75, 255, 255, 255, 255 );
316         screen->SimpleLine( 190, 75, 50, 75, 255, 255, 255, 255 );
317
318         screen->Circle( &temp, &Taste1, 5, true );
319         screen->SimpleText( std::string( "T0" ), (int)Taste1.GetX( )-15,
(int)Taste1.GetY( ), 0, 0, 255, 127 );
320
321         screen->Circle( &temp, &Taste2, 5, true );
322         screen->SimpleText( std::string( "T1" ), (int)Taste2.GetX( )-15,
```

```
(int)Taste2.GetY( ) , 0, 0, 255, 127 );
323
324
325 }
326
327 void DrawPBSolution( SDL::SDL_Graphic *screen )
328 {
329     SDL::color Taste1( 50, 75 );
330     SDL::color Taste2( 50, 190 );
331
332     SDL::color temp( 255, 255, 255, 255 );
333     SDL::color temp0( 0, 255, 0, 255 );
334     SDL::color temp1( 255, 0, 0, 255 );
335     SDL::color temp2( 0, 0, 255, 255 );
336
337     L[0].DrawMe( screen );
338     screen->SimpleText( std::string( "L1" ), (int)L[0].GetCoor( )->GetX( ),
(int)L[0].GetCoor( )->GetY( ), 0, 0, 255, 127 );
339     L[1].DrawMe( screen );
340     screen->SimpleText( std::string( "L2" ), (int)L[1].GetCoor( )->GetX( )
, (int) L[1].GetCoor( )->GetY( ), 0, 0, 255, 127 );
341     L[2].DrawMe( screen );
342     screen->SimpleText( std::string( "L3" ), (int)L[2].GetCoor( )->GetX( ),
(int)L[2].GetCoor( )->GetY( ), 0, 0, 255, 127 );
343
344     screen->SimpleLine( 50, 75, 50, 190, 255, 255, 255, 255 );
345     screen->SimpleLine( 50, 190, 190, 190, 255, 255, 255, 255 );
346     screen->SimpleLine( 190, 190, 190, 75, 255, 255, 255, 255 );
347     screen->SimpleLine( 190, 75, 50, 75, 255, 255, 255, 255 );
348
349
350     S[0].DrawMe( screen, (zustand == 0) ? true : false );
351     screen->SimpleText( std::string( "Z1" ), (int)S[0].GetCoor( )->GetX( ),
(int)S[0].GetCoor( )->GetY( ), 0, 0, 255, 127 );
352     S[1].DrawMe( screen, (zustand == 1) ? true : false );
353     screen->SimpleText( std::string( "Z2" ), (int)S[1].GetCoor( )->GetX( ),
(int)S[1].GetCoor( )->GetY( ), 0, 0, 255, 127 );
354     S[2].DrawMe( screen, (zustand == 2) ? true : false );
355     screen->SimpleText( std::string( "Z3" ), (int) S[2].GetCoor( )->GetX( ),
(int)S[2].GetCoor( )->GetY( ), 0, 0, 255, 127 );
356
357     S[0].DrawLines( screen );
358     S[1].DrawLines( screen );
359     S[2].DrawLines( screen );
360
361     screen->Circle( &temp, &Taste1, 5, true );
362     screen->SimpleText( std::string( "T0" ), (int)Taste1.GetX( )-15,
(int)Taste1.GetY( )-10 , 0, 0, 255, 127 );
363
364     screen->Line( &temp2, &Taste1, S[ keys[0][ keys[0][ keys[0][zustand]
] ] ].GetCoor( ) );
365     screen->Line( &temp1, &Taste1, S[ keys[0][ keys[0][zustand] ] ]
.GetCoor( ) );
366     screen->Line( &temp0, &Taste1, S[ keys[0][zustand] ].GetCoor( ) );
367
368     screen->Circle( &temp, &Taste2, 5, true );
369     screen->SimpleText( std::string( "T1" ), (int)Taste2.GetX( )-15,
(int)Taste2.GetY( )-10 , 0, 0, 255, 127 );
369
370     screen->Line( &temp2, &Taste2, S[ keys[1][ keys[1][ keys[1][zustand]
] ] ].GetCoor( ) );
371     screen->Line( &temp1, &Taste2, S[ keys[1][ keys[1][zustand] ] ]
.GetCoor( ) );
372     screen->Line( &temp0, &Taste2, S[ keys[1][zustand] ].GetCoor( ) );
372
```

```
373
374
375
376
377     }
378
379
380 };
381 }
```

main.cpp

```
1  /*
2   PaderBox BW 2006 by Alexander Weld
3   Part of the BWINF25-Solution by team "Die Schnabeltiere featuring Hr. Höllwerber
4       powered by knoob.de"
5
6   main.cpp
7
8   V0.0 11.11.2006
9   V0.1 11.11.2006
10  V0.2 12.11.2006 Graphics -- SDL
11 */
12 #include <cstdlib>
13 #include <time.h>
14 #include <iostream>
15 #include <string>
16 #include <sstream>
17 #include <cstdio>
18 #include "main.h"
19 // #define DEBUG_KEYS
20 // #define DEBUG_VIEW
21 using namespace std;
22 using namespace PB;
23 int main(int argc, char *argv[])
24 {
25     SDL::Init( );
26
27     SDL::SDL_Graphic *screen;
28
29     PaderBox *pb;
30
31     bool mainloop = true;
32     bool solutionloop = true;
33
34     string temp;
35
36     //char c;
37
38     SDL_Event event;
39
40     srand( time( NULL ) );
41
42     screen = new SDL::SDL_Graphic( );
43
44     pb = new PaderBox( );
45
46     while( mainloop )
47     {
48         #ifdef DEBUG_VIEW
49             pb->PrintMatrix();
50             pb->PrintKeys();
51             cout << "Zustand: " << pb->GetZustand() << endl;
52         #endif
53     }
```

```
53         //Graphic:
54
55         screen->Clear( );
56
57         screen->SimpleText( string( "PaderBox BW 2006 by Alexander Weld" ),
58         5, 5, 255, 255, 255, 255 );
59
60         screen->SimpleText( string( "Part of the BWINF25-Solution by team
61         \"Die Schnabeltiere featuring Hr. Höllwerber powered by knoob.de\" ), 5,
62         15, 255, 255, 255, 255 );
63
64         screen->SimpleText( string( "Hilfe:" ), 500, 30, 255, 255, 0, 255 );
65         screen->SimpleText( string( "Taste 0 drücken: '0'" ), 500, 40, 255,
66         255, 255, 255 );
67         screen->SimpleText( string( "Taste 1 drücken: '1'" ), 500, 50, 255,
68         255, 255, 255 );
69         screen->SimpleText( string( "Lösung anzeigen: 's'" ), 500, 60, 255,
70         255, 255, 255 );
71         screen->SimpleText( string( "Zurücksetzen: 'z'" ), 500, 70, 255,
72         255, 255, 255 );
73         screen->SimpleText( string( "Beenden: 'x'" ), 500, 80, 255, 255,
74         255, 255 );
75
76         if( solutionloop )
77             pb->DrawPB( screen );
78         else
79             {
80                 screen->SimpleText( string( "Erklärung:" ), 500, 100, 255, 255,
81                 0, 255 );
82                 screen->SimpleText( string( "Roter Zustand: Aktueller Zustand" ),
83                 500, 110, 255, 255, 127, 255 );
84                 screen->SimpleText( string( "Verbindung vom Schalter zum
85                 Zustand:" ), 500, 120, 255, 255, 127, 255 );
86                 screen->SimpleText( string( "Grün: Nächster Zustand" ), 500, 130,
87                 255, 255, 127, 255 );
88                 screen->SimpleText( string( "Rot: Zustand nach 'Grün'" ), 500,
89                 140, 255, 255, 127, 255 );
90                 screen->SimpleText( string( "Blau: Zustand nach 'Rot'" ), 500,
91                 150, 255, 255, 127, 255 );
92                 pb->DrawPBSolution( screen );
93             }
94
95         screen->Flip( );
96
97         SDL_WaitEvent( &event );
98
99         switch( event.type )
100         {
101             case SDL_KEYDOWN:
102                 switch( event.key.keysym.sym )
103                 {
104                     case SDLK_0:
105                         pb->KeyPress( 0 );
106                         break;
107                     case SDLK_1:
108                         pb->KeyPress( 1 );
109                         break;
110                     case SDLK_x:
111                         mainloop = false;
112                         break;
113                     case SDLK_s:
114                         solutionloop = !solutionloop;
115                         break;
116                     case SDLK_y:
117                         pb->Reset( );
118                         break;
119                     /*case SDLK_1:
```

```
105             break;
106         case SDLK_1:
107             break;*/
108         default:
109             break;
110
111             }
112         }
113
114     }
115 }
116
117
118
119 delete screen;
120
121 SDL::DeInit( );
122
123 cout << "PaderBox BW 2006 by Alexander Weld" << endl
124      << "Part of the BWINF25-Solution by team \"Die Schnabeltiere featuring
125      Hr. Höllwerber powered by knoob.de\"" << endl;
126 //system("PAUSE");
127 return EXIT_SUCCESS;
128 }
```