1. Lösungsidee

Dazu lässt sich eigentlich nur sagen, dass die Angaben 1:1 umgesetzt werden sollen.
Es gibt nur 2 Dinge, die geklärt werden müssen:
1. Wie wird ein ganzzahliger Zufallswert zwischen 0 und r_max am besten auf eine ganze Zahl zwischen min und max (beide eingeschlossen) abgebildet?
2. Wie verbindet man die Lampen mit den Zuständen und umgekehrt?

Zu 1:
Am einfachsten ist der Modulo-Ansatz: r % (max - min + 1) + min, der auch, wie man mit TestGetRandom() sehen kann, eine beinahe perfekte Verteilung bietet.

Zu 2:
Jede Lampe ist mit genau 2 Zuständen verbunden und jeder Zustand ist mit genau 2 Lampen verbunden, es ist also eine bidirektionale Verbindung und es reicht für alle Zustände anzugeben mit welchen Lampen sie verbunden sind, um die Umkehrung zu bestimmen.
Der erste Zustand kann sich aus den 3 Lampen frei 2 aussuchen, der zweite dagegen muss die andere, noch freie Lampe wählen und dann eine von den beiden Lampen des 1. Zustandes, da sonst der 3. Zustand nur 1 Lampe zur Auswahl hätte. Der 3. Zustand muss dann einfach die letzten beiden freien Verbindungen „wählen“, wobei er die eine Verbindung mit dem 1., die andere mit dem 2. Zustand gemein hat.
Beispiel, wieso der 2. Zustand nur 1 Lampe frei wählen darf:
Zustand 1 -> Lampe 1, Lampe 2
Zustand 2 -> Lampe 1, Lampe 2 (!)
=> Zustand 3 -> Lampe 3 (Widerspruch: es kann keine 2. Verbindung gefunden werden!)
Beispiel für das oben beschriebenes Vorgehen:
Zustand 1 -> Lampe 1, Lampe 3
Zustand 2 -> Lampe 1 oder 3, Lampe 2
Zustand 3 -> Lampe 3 oder 1, Lampe 2

2. Programm-Dokumentation

Das Programm besteht eigentlich aus 2 Einheiten: einerseits der Paderbox-Klasse, die das Verhalten einer Paderbox beschreibt und die zufällige Initialisierung durchführt, andererseits das Hauptprogramm, das dem Benutzer simple Konsolen-Menüs zur Verfügung stellt um das Spiel zu starten und die Paderbox zu bedienen.
Das einzige Interessante ist die Initialisierung, die aber genauso umgesetzt wird, wie in der Lösungsidee beschrieben aber etwas gestraffter ist.
Für die Wahl der zweiten Lampe des 1. Zustandes, verwende ich folgenden Code:

```
do {
      stateLampLinks[ 0 ][ 1 ] = GetRandom( 0, LAMP_COUNT - 1 );
} while( stateLampLinks[ 0 ][ 0 ] == stateLampLinks[ 0 ][ 1 ] );
```

Die Frage ist, ob durch die Schleife die Wahrscheinlichkeit der gewählten Lampe irgendwie beeinflusst wird, die 1/2 betragen sollte, da ja eine Lampe wegfällt.
Die Wahrscheinlichkeit, die sich aus dem Quellcode ergibt, beträgt:

$$\frac{1}{2}+\frac{1}{3}\cdot\frac{1}{2}+(\frac{1}{3})^2\cdot\frac{1}{2}+\ldots=\lim_{n\to\infty}\sum_{i=0}^{n}\frac{1}{2}(\frac{1}{3})^i=\frac{1}{2}\lim_{n\to\infty}\sum_{i=0}^{n}(\frac{1}{3})^i=\frac{1}{2}\cdot\frac{1}{1-\frac{1}{3}}=\frac{1}{2}\cdot\frac{3}{2}=\frac{1}{2}$$

und ist damit korrekt.

Das Programm ist eine einfache Konsolen-Applikation, erlaubt aber trotzdem das Tüfteln an der Geheimprogrammierung, und ist möglicherweise sogar übersichtlicher als eine grafische Anwendung, da man immer die alten Lampenzustände vor Augen hat und sich dadurch weniger merken muss.

3. Programm-Ablaufprotokoll

```
[...]>"Aufgabe 5.exe"

P Starte ein Spiel
Q Ende des Programms

Menüwahl: p
Spiel startet..

Lampe 1: AN Lampe 2: AN Lampe 3: AN
0 Drücke Taste 0
1 Drücke Taste 1
R Setze die PaderBox zurück
? Verrate die Programmierung
Q Gehe zurück zum Hauptmenü

Menüwahl: 0
Lampe 1: AUS Lampe 2: AUS Lampe 3: AN
0 Drücke Taste 0
1 Drücke Taste 1
R Setze die PaderBox zurück
? Verrate die Programmierung
Q Gehe zurück zum Hauptmenü

Menüwahl: 0
Lampe 1: AN Lampe 2: AN Lampe 3: AN
0 Drücke Taste 0
1 Drücke Taste 1
R Setze die PaderBox zurück
? Verrate die Programmierung
Q Gehe zurück zum Hauptmenü

Menüwahl: 1
Lampe 1: AUS Lampe 2: AN Lampe 3: AUS
0 Drücke Taste 0
1 Drücke Taste 1
R Setze die PaderBox zurück
? Verrate die Programmierung
Q Gehe zurück zum Hauptmenü

Menüwahl: r
Lampe 1: AN Lampe 2: AN Lampe 3: AN
0 Drücke Taste 0
1 Drücke Taste 1
R Setze die PaderBox zurück
? Verrate die Programmierung
Q Gehe zurück zum Hauptmenü

Menüwahl: ?
Taste 0         Taste 1 (Z steht für Zustand)
Z1->Z1          Z1->Z3
Z2->Z1          Z2->Z1
Z3->Z1          Z3->Z2

Zustand         Lampen
Z1              Lampe 2, Lampe 1
Z2              Lampe 2, Lampe 3
Z3              Lampe 1, Lampe 3

Anfangszustand: Z1
```

```
Lampe 1: AN Lampe 2: AN Lampe 3: AN
0 Drücke Taste 0
1 Drücke Taste 1
R Setze die PaderBox zurück
? Verrate die Programmierung
Q Gehe zurück zum Hauptmenü

Menüwahl: q

P Starte ein Spiel
Q Ende des Programms

Menüwahl: q
```

4. Quellcode

```
#include <iostream>
#include <cstdlib>
#include <assert.h>
#include <ctime>

using namespace std;

unsigned GetRandom( unsigned min, unsigned max ) {
	assert( max - min + 1 < RAND_MAX );
	return rand() % (max - min + 1) + min;
}
/*
void TestGetRandom() {
	const unsigned width = 4, attempts = width * 10000000;
	unsigned hits[ width + 1] = { 0 };
	for( unsigned i = 0 ; i < attempts ; i++ ) {
		unsigned randomNumber = GetRandom( 1, width );
		hits[ randomNumber - 1 ]++;
	}
	for( i = 0 ; i < width + 1; i++ )
		cout << hits[ i ] << " ";
	cout << endl;
	double E = 0.0;
	for( i = 0 ; i < width ; i++ )
		E += (double) (i+1) * hits[ i ] / attempts;
	cout << "E = " << E << endl;
	double Esquared = 0.0;
	for( i = 0 ; i < width ; i++ )
		Esquared += (double) (i+1) * (i+1) * hits[ i ] / attempts;
```

```
	double Var = Esquared - E*E;
	cout << "Var = " << Var << endl;
}
void main() {
	TestGetRandom();
}
*/

class PaderBox {
public:
	enum Buttons {
		BUTTON_0,
		BUTTON_1,
		BUTTON_COUNT
	};
private:
	static const unsigned LAMP_COUNT = 3;
	static const unsigned STATE_COUNT = 3;
	static const unsigned LINK_COUNT = 2;
	unsigned stateLampLinks[ STATE_COUNT ][ LINK_COUNT ];
	// [button index][old state]
	unsigned stateTransitionMap[ BUTTON_COUNT ][ STATE_COUNT ];

	struct State {
		bool lampState[ LAMP_COUNT ];
		unsigned activeState;
	} initialState, currentState;

private:
	void SetupTransitions() {
		// setup the state transition map
		for( unsigned button = 0 ; button < BUTTON_COUNT ; button++ ) {
			for( unsigned state = 0 ; state < STATE_COUNT ; state++ ) {
				stateTransitionMap[ button ][ state ] = GetRandom( 0, STATE_COUNT - 1 );
			}
		}
		// possibilities: 3^(2*3)=3^6=729
	}

	void SetupLampLinks() {
```

```
		// the first state can choose the two lamps freely (thus 3*2/2*1 possibilities)
		// the second state can choose the first lamp freely but:
		// if an already linked lamp is chosen, then it must choose the unlinked lamp next
		// if the unlinked lamp is chosen, it can choose between two linked lamps next
		// the third state now has to choose the two links that are left
		// (possibilities: 3*2/2*1 * (2*1+1*2)/2 * 1 = 3 * 2 * 1 = 6)
		assert( LAMP_COUNT == 3 && STATE_COUNT == 3 );
		stateLampLinks[ 0 ][ 0 ] = GetRandom( 0, LAMP_COUNT - 1 );
		do {
			stateLampLinks[ 0 ][ 1 ] = GetRandom( 0, LAMP_COUNT - 1 );
		} while( stateLampLinks[ 0 ][ 0 ] == stateLampLinks[ 0 ][ 1 ] );

		if( GetRandom( 0, 1 ) ) {
			stateLampLinks[ 1 ][ 0 ] = stateLampLinks[ 0 ][ 0 ];
			stateLampLinks[ 2 ][ 0 ] = stateLampLinks[ 0 ][ 1 ];
		} else {
			stateLampLinks[ 1 ][ 0 ] = stateLampLinks[ 0 ][ 1 ];
			stateLampLinks[ 2 ][ 0 ] = stateLampLinks[ 0 ][ 0 ];
		}
		stateLampLinks[ 1 ][ 1 ] = stateLampLinks[ 2 ][ 1 ] = 0+1+2 - (stateLampLinks[ 0 ][ 0 ] +
stateLampLinks[ 0 ][ 1 ]);
		// possibilities using this algorithm: 3*2/2*2 = 6
	}

	void SetupInitialState() {
		initialState.activeState = GetRandom( 0, STATE_COUNT - 1 );
		for( unsigned i = 0 ; i < LAMP_COUNT ; i++ ) {
			initialState.lampState[ i ] = GetRandom( 0, 1 ) == 1;
		}
		// possibilities: 3 * 2^3 = 24
	}

public:
	PaderBox() {
		SetupTransitions();
		SetupLampLinks();
		SetupInitialState();
		Reset();
	}
```

```
	void ChangeState( unsigned newState ) {
		currentState.activeState = newState;
		for( unsigned i = 0 ; i < LINK_COUNT ; i++ ) {
			const unsigned lampIndex = stateLampLinks[ newState ][ i ];
			bool &lampState = currentState.lampState[ lampIndex ];
			lampState = !lampState;
		}
	}

public:
	void Reset() {
		currentState = initialState;
	}

	void PressButton( const Buttons button ) {
		ChangeState( stateTransitionMap[ button ][ currentState.activeState ] );
	}

	void DisplayLampState() const {
		for( unsigned i = 0 ; i < LAMP_COUNT ; i++ ) {
			cout << "Lampe " << (i+1) << ": " << (currentState.lampState[ i ] ? "AN" : "AUS") << " ";
		}
		cout << endl;
	}

	void DisplaySecretProgramming() const {
		assert( BUTTON_COUNT == 2 );
		cout << "Taste 0\t\tTaste 1 (Z steht für Zustand)" << endl;
		for( unsigned i = 0 ; i < STATE_COUNT ; i++ ) {
			cout << "Z" << (i + 1) << "->Z" << stateTransitionMap[ 0 ][ i ] + 1 << "\t\tZ" << (i + 1) <<
"->Z" << stateTransitionMap[ 1 ][ i ] + 1 << endl;
		}
		cout << endl;

		assert( LINK_COUNT == 2 );
		cout << "Zustand\t\tLampen" << endl;
		for( i = 0 ; i < STATE_COUNT ; i++ ) {
			cout << "Z" << (i + 1) << "\t\tLampe " << stateLampLinks[ i ][ 0 ] + 1 << ", Lampe " <<
stateLampLinks[ i ][ 1 ] + 1 << endl;
		}
```

```
		cout << endl;

		cout << "Anfangszustand: Z" << initialState.activeState + 1 << endl;
	}
};

void RunGame() {
	PaderBox paderBox;
	cout << "Spiel startet.." << endl << endl;

	bool endGame = false;
	while( !endGame ) {
		paderBox.DisplayLampState();

		while( 1 ) {
			cout << "0 Drücke Taste 0" << endl
				<< "1 Drücke Taste 1" << endl
				<< "R Setze die PaderBox zurück" << endl
				<< "? Verrate die Programmierung" << endl
				<< "Q Gehe zurück zum Hauptmenü" << endl
				<< endl;
			cout << "Menüwahl: ";
			unsigned char choice;
			cin >> choice;

			switch( choice ) {
				case '0':
					paderBox.PressButton( PaderBox::BUTTON_0 );
					break;
				case '1':
					paderBox.PressButton( PaderBox::BUTTON_1 );
					break;
				case 'R':
				case 'r':
					paderBox.Reset();
					break;
				case '?':
					paderBox.DisplaySecretProgramming();
					break;
				case 'Q':
```

```
                                case 'q':
                                        return;
                                        break;
                                default:
                                        continue;
                        }
                        break;
                }
        }
}

void ShowMainMenu() {
        bool quit = false;
        while( !quit ) {
                cout << endl
                        << "P Starte ein Spiel" << endl
                        << "Q Ende des Programms" << endl
                        << endl;
                cout << "Menüwahl: ";
                unsigned char choice;
                cin >> choice;
                switch( choice ) {
                        case 'P':
                        case 'p':
                                RunGame();
                                break;
                        case 'Q':
                        case 'q':
                                quit = true;
                                break;
                        default:
                                break;
                }
        }
}

void main() {
        srand( time( NULL ) );
        ShowMainMenu();
}
```