Tipps zur Spieleprogrammierung in C

von Björn Flintrop und Irene Rothe

Zufallszahlen:

Bei vielen Spielen werden Zufallszahlen benötigt, z.B. für ein Würfelergebnis oder ähnliches.

Lösung:

1. Fügen Sie oben in Ihrem Programm folgende Bibliotheken hinzu:

   ```
   #include <stdlib.h>
   #include <time.h>
   ```

2. Fügen Sie folgenden Code am Anfang des main-Anweisungsblocks hinzu:

   ```
   srand ( time(NULL) );
   ```

3. Deklarieren Sie eine Variable als Speicher für eine Zufallszahl, z.B.:

   ```
   int zufallszahl;
   ```

4. Mit dem folgenden Code wird eine Zufallszahl zwischen 0 und max-1 in die Variable geschrieben (max steht hier für 'Maximum'):

   ```
   zufallszahl = rand() % max;
   ```

Verwenden von Farben:

Grundsätzlich ist es auch möglich, die Farbe des auszugebenen Textes zu verändern. Es stehen insgesamt 16 Farben zur Verfügung.

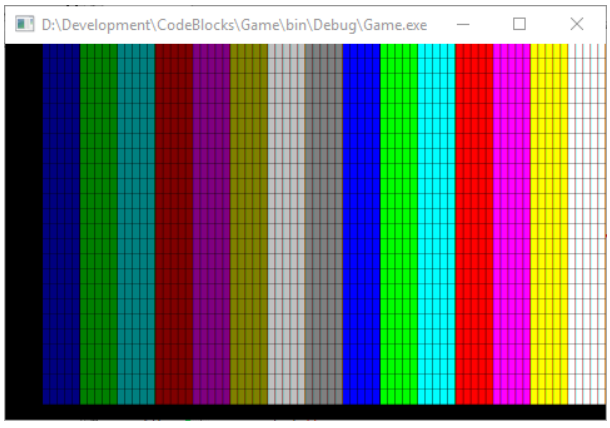

Abb. 1: die 16 möglichen Schriftfarben

Linux etc.: Der folgende Befehl erzeugt z.B. die so genannte 'Escape-Sequenz' zum Einstellen der Schriftfarbe Rot (Wert: 31):

```
printf("\e[31m");
```

Das funktioniert auf **Linux** und **MacOS** Systemen (und in **onlinegdb**).

Windows: hier muss ein anderer Ansatz gewählt werden:

1. Fügen Sie oben in Ihrem Programm folgende Bibliothek hinzu:

   ```
   #include <windows.h>
   ```

2. Fügen Sie folgende Zeilen am Anfang des main-Anweisungsblocks hinzu:

   ```
   HANDLE h = GetStdHandle ( STD_OUTPUT_HANDLE );
   ```

3. Einstellen der Schriftfarbe z.B. auf Rot (Wert 4):

   ```
   SetConsoleTextAttribute (h, 4);
   ```

Tabelle der Farbcodes Linux/Windows:

Farbe	Linux	Windows
Black	30	0
Red	31	4
Green	32	2
Yellow	33	6
Blue	34	1
Magenta	35	5
Cyan	36	3
Light Gray	37	7
Dark Gray	90	8
Light Red	91	12
Light Green	92	10
Light Yellow	93	14
Light Blue	94	9
Light Magenta	95	13
Light Cyan	96	11
White	97	15

Eingabe beliebiger, einzelner Zeichen:

Falls mit **scanf** ein Zeichen (Platzhalter: %c) eingegeben werden soll, so sollte ein Leerzeichen vor den Platzhalter gesetzt werden. Also z.B.:

```
char zeichen;

scanf(" %c", &zeichen);
```

Eingabe ohne ENTER-Taste:

Die Funktion ***scanf*** gibt die Eingabe erst nach Betätigung der Enter-Taste an das Programm. Falls sofort auf einen Tastendruck reagiert werden soll, kann dies so realisiert werden:

Es muss die C-Programmbibliothek conio.h (Windows) oder ncurses.h (Linux) eingebunden werden, die entsprechende Funktionen zum Abfragen eines Tastendrucks bereitstellt. Interessant sind in diesem Zusammenhang vor allem *getch()* und *kbhit()*:

```
#include <conio.h>

// Wartet auf einen Tastendruck und
// speichert den ASCII-Wert der Taste.

int taste;
taste = getch();

// Falls eine Taste gedrueckt ist, wird der Wert gespeichert.
// Diese Zeile wartet nicht!

if(kbhit())
{
   taste = getch();
}
```

Mischen:

Bei der Programmierung von Spielen kann es erforderlich sein, **Spielkarten** oder ähnliche Dinge zu mischen.

Tipp zum Mischen:

```
int karten[10] = {1,2,3,4,5,6,7,8,9,10};

int i, j;

srand( time(NULL) );            // Start Zufallsgenerator

for(i=0; i<10; i++)
{
    int hilfskarte;

    j = rand() % 10;            // Zufallszahl zwischen 0 und 9

    hilfskarte = karten[i];

    karten[i] = karten[j];

    karten[j] = hilfskarte;
}
```

Für Fortgeschrittene:

Die Heranführung an die Programmiersprache C bzw. die Durchführung der ersten Programmieraufgaben in dieser Sprache erfolgt üblicherweise in einer rein Tastatur-basierten Form. Es können Werte oder Zeichenketten per ***scanf*** eingegeben und per ***printf*** ausgegeben werden. Eine Eingabe muss hierbei stets mit einem Drücken der Enter-Taste abgeschlossen werden. Die Ausgaben erfolgen in der Konsole bzw. dem Terminal (in Windows auch Eingabeaufforderung, DOS-Fenster oder cmd.exe genannt) einfach hintereinander, so als ob man Sie immer wieder an das Ende eines Textdokuments kopieren würde.

Problem: Bei der Programmierung von Spielen kommt man hier schnell an gewisse Grenzen, vor allem wenn es um die Ausgabe einer Art von ‚Spielfeld' geht oder wenn bestimmte Tastatureingaben ohne ‚Enter' eine Änderung des Spiels bewirken sollen. Beispiele hierfür sind zum Beispiel die Spiele Snake und Pong.

Abb. 2: Snake im Textmodus

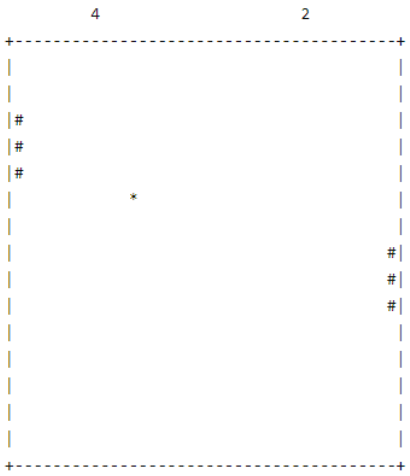

Abb 3: Pong im Textmodus

Tipps zur Ausgabe: Falls, wie bei Snake oder Pong, ein ganzes Spielfeld ausgegeben werden soll, dessen Position sich im Terminalfenster nicht ändert, so wird typischerweise der komplette Fensterbereich genutzt. Die Standardgröße der Ausgabe beträgt 25 Zeilen mit je 80 Spalten (Zeichen). Es bietet sich an hierfür ein zweidimensionales Array vom Datentyp char zu verwenden, z.B.:

```
char screen[25][81];
```

Da in C Zeichenketten als Arrays vom Datentyp char realisiert werden, entspricht die Variable *screen* in diesem Beispiel ganz einfach den 25 Zeichenketten, die für die Füllung der 25 Zeilen benötigt werden.

Mittels printf("%s\n", screen[y]); kann eine Zeile y ausgegeben werden.

Aber warum jetzt 81 Zeichen pro Zeile? In C müssen Zeichenketten immer mit einem Null-Byte beendet werden. Dieses eine Byte kommt also zu den 80 Zeichen der Zeile hinzu.

Gibt man nun in jedem Spielschritt alle 25 Zeilen aus, so wird immer das komplette Fenster neu gefüllt und es entsteht der Eindruck eines ‚statischen' Spielfelds.

Flackern bei Aktualisierung des Spielfeldes:

Da die 25 Zeilen ja eigentlich immer wieder unten angefügt werden, und dann (wenn auch sehr schnell) nach oben wandern, kann es vorkommen, dass eine Art Ruckeln in der Anzeige auftritt.

Lösung für Linux, MacOS (und onlinegdb):

Es gibt auch eine Escape-Sequenz zum Löschen des Fensterinhalts, welche folgendermaßen ausgegeben werden kann:

```
printf("\e[2J");
```

Durch einen Aufruf dieses Befehls vor der Ausgabe des Spielfelds kann ein Flackern vermieden werden.

Lösung für Windows:

1. Fügen Sie oben in Ihrem Programm folgende Bibliothek hinzu:

   ```
   #include <windows.h>
   ```

2. Fügen Sie folgende Zeilen am Anfang des main-Anweisungsblocks hinzu:

   ```
   HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
   COORD coord;
   coord.X = 0;
   coord.Y = 0;
   ```

3. Und dann immer vor der Ausgabe der neuen 25 Zeilen:

   ```
   SetConsoleCursorPosition(h, coord);
   ```

Programmablauf ohne Tastatureingabe:

Spiele wie Pong oder Snake laufen, einmal gestartet, auch ohne Tastatureingaben solange weiter, bis das aktuelle Spiel beendet ist. Hierbei wird eine bestimmte Spielgeschwindigkeit eingehalten, d.h. es wird x mal pro Sekunde ein neuer Spielschritt berechnet. Um so ein Verhalten zu realisieren, muss zwischen den Spielschritten eine bestimmte Zeitspanne gewartet werden. Hierfür stellt die Bibliothek *time* die Funktion *usleep* zur Verfügung, die das Programm für eine bestimmte Anzahl von Mikrosekunden anhält.

```
#include <time.h>

int game_over = 0;

do
{
   game_over = perform_game_step();
   usleep(200000);
}
while(!game_over);
```

Die Schleife im obigen Beispiel wartet bei jedem Durchgang für 200000 Mikrosekunden, was 200 Millisekunden entspricht. Vernachlässigt man die Rechenzeit für die Ausführung von *perform_game_step*, kann man somit erwarten, dass die Schleife 5 mal pro Sekunde durchlaufen wird. Es wird also 5 mal pro Sekunde ein neuer Spielschritt berechnet.