|  |  |
| --- | --- |
| http://carme.bfh.ch/libBSPdoc/Mainpage.png | http://www.serenabasciani.it/blog/nokia-snake-cheevo-530w.jpg |

Carme-Snake

# Ein Programmierprojekt in C und Assembler

Hardwarenahe Softwareentwicklung  
18. Januar 2013

Carme-Snake Projekt von

Cyril Stoller   
Marcel Bärtschi  
Peter Ambühl

Dozent

R. Weber

Inhalt

[Ein Programmierprojekt in C und Assembler 1](#_Toc346412359)

[Übersicht 3](#_Toc346412360)

[Modulaufteilung 4](#_Toc346412361)

[PC Client Code 4](#_Toc346412362)

[Spiel starten 5](#_Toc346412363)

[Hall of Fame 6](#_Toc346412364)

[Comport wählen 7](#_Toc346412365)

[Microcontroller Code 8](#_Toc346412366)

[Interrupt Handler 8](#_Toc346412367)

[main 9](#_Toc346412368)

[snake\_controller 10](#_Toc346412369)

[Graphics 11](#_Toc346412370)

[Spielbeschrieb 12](#_Toc346412371)

[Bedienungsanleitung 12](#_Toc346412372)

[Mögliche Erweiterungen 13](#_Toc346412373)

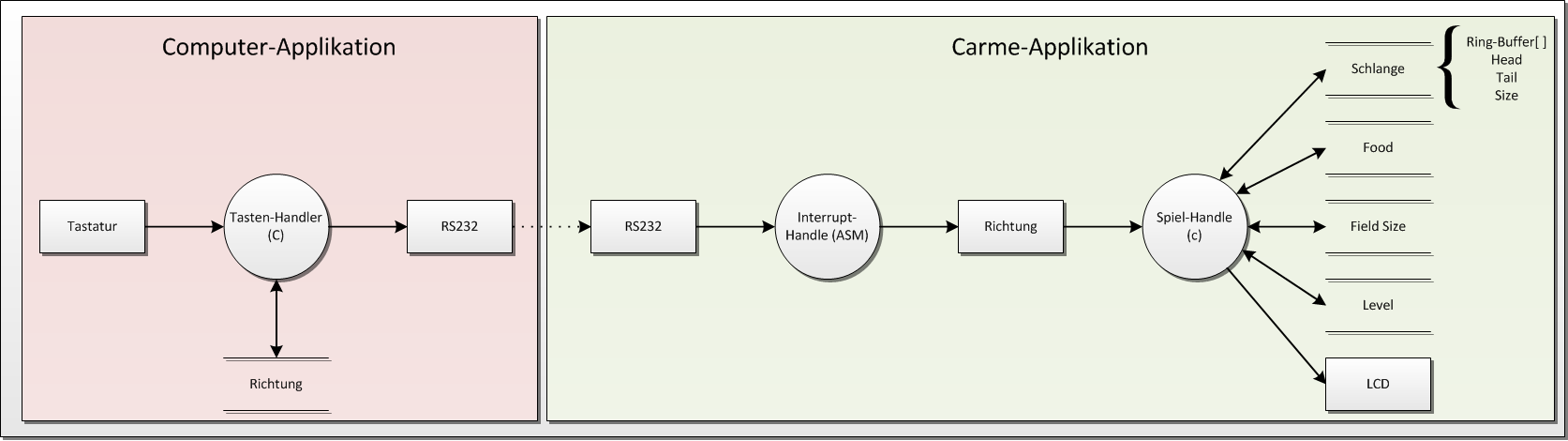
[Fazit 13](#_Toc346412374)

[Source Code Hosting 13](#_Toc346412375)

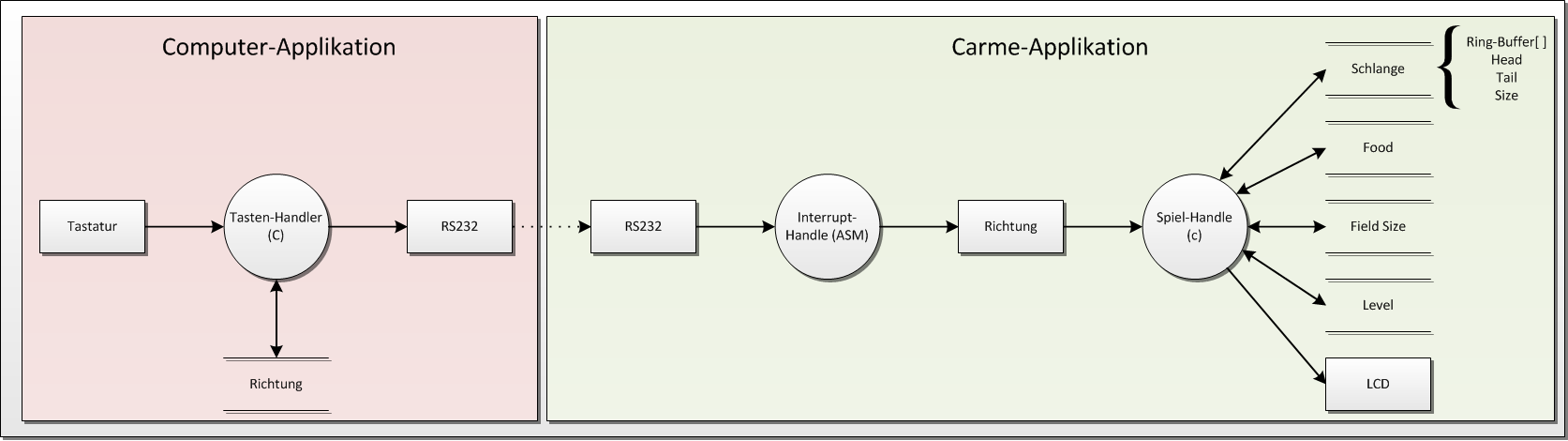
[Weiterführende Dokumentation mit Doxygen 13](#_Toc346412376)

# Übersicht

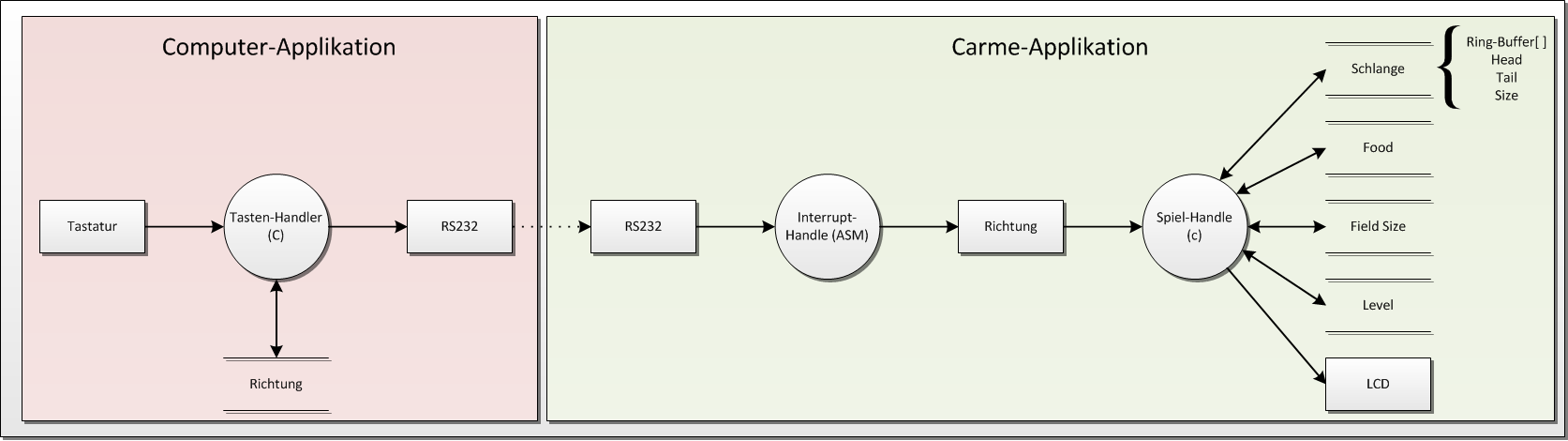
Das Spiel „Snake“ erlangte vor allem durch seine Implementation auf Mobilfunkgeräten von Nokia in den 1990er Jahren Bekanntheit. Wir haben uns zum Ziel gesetzt, dieses auf dem Carme-Kit nachzubauen.



Zur Darstellung und allgemeinen logischen Spielsteuerung wird der ARM-basierte PXA270 und das 320x240 Pixel Farb-LCD des Carme-Kits verwendet. Die Bedienung erfolgt mangels geeigneter Pfeiltasten des Carme-Kits auf einer Computertastatur. Von dort werden die Tastenbefehle per RS232 Verbindung an das Carme-Kit gesendet.



Der Computer ist vor allem für das Einlesen der Pfeiltasten zuständig. Die Softwarebesitzt aber auch eine GUI Oberfläche mit welcher die Highscores angeschaut und der Comport gewählt werden kann.



Das Carme Kit empfängt diese Daten und kontrolliert damit das Spiel. Es ist ausserdem auch für die graphische Darstellung der Schlange zuständig.

# Modulaufteilung

Ein Modul ist der PC Client, welcher Pfeiltasten sammelt und weiterleitet. Dieses Modul ist gleichzeitig auch eine ausführbare Datei.

Ein weiteres Modul ist in Assembler geschrieben und behandelt alle hardwarenahen Funktionen des Carme-Kits. Es empfängt Pfeiltasten über UART0 und aktualisiert die entsprechende Richtung in einer globalen Variablen. Es sendet die erreichte Punktzahl über UART0 und kümmert sich ausserdem um den Timer-Interrupt, welcher für die Zeitverzögerungsfunktion gebraucht wird.

Dann gibt es noch drei Module, welche sich um das eigentliche Spiel kümmern: Das main-Modul, das graphics-Modul und snake\_controller-Modul. Es gibt ebenfalls noch das Modul marsenne, welches eine Bibliothek enthält um Pseudozufallszahlen zu generieren. Es wurde unbearbeitet in das Projekt eingebunden und deshalb auch nicht dokumentiert. Die Dokumentation kann hier gefunden werden: <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

# PC Client Code

Die Software auf dem PC läuft in C mit der GUI Library von Ivo Oesch. Das GUI stellt auf dem Hauptmenü vier Menüpunkte bereit:



In einer Endlosschlaufe wird auf eine gedrückte Taste gewartet und dessen Menüfunktion ausgeführt. Drückt man ESC oder Alt+F4 oder wählt den Menüpunkt 4, wird das Programm beendet.



## Spiel starten

Zuerst muss hier der Spielername eingegeben und mit **Enter** bestätigt werden. Dann werden Pfeiltasten eingelesen und diese über den mit Menüpunkt 4 gewählten Comport über RS232 gesendet. Es wird jeweils eine Pfeiltaste gesendet, wenn sie unterschiedlich als die vorherige ist. Zudem werden Eingaben, die „rückwärts“ zur aktuellen Richtung sind, ignoriert.

Gesendet wird entweder ein ‚r‘ für *rechts*, ein ‚u‘ (up) für *auf*, ein ‚l‘ für *links* oder ein ‚d‘ (down) für *ab*.

Empfängt der PC die Score über RS232 vom Carme-Kit, wird diese in der Highscoreliste eingetragen und überprüft, ob die aktuelle Score die höchste ist.



## Hall of Fame

Hier wird das File „highscore.txt“ im Ausführungsordner eingelesen und dessen Einträge dargestellt. Ist das File nicht im Ordner enthalten, wird es neu erstellt. Das File kann auch ohne das Spiel zu starten gelesen werden, dies ermöglicht die einfache Verwendung der Daten. Jedoch gehen wir davon aus, dass das File nicht manuell modifiziert wird, wir sind ja ehrenhafte Spieler. Nach fünf Sekunden gelangt man automatisch wieder in das Hauptmenü.

## Comport wählen

Hier wird einfach ein einziges Zeichen eingelesen. Wenn das Zeichen eine Zahl zwischen 1 und 7 war, wird die Comport-Nummer auf diese Zahl geändert, ansonsten und sofort nach Programmstart gilt der Default-Comport COM4.

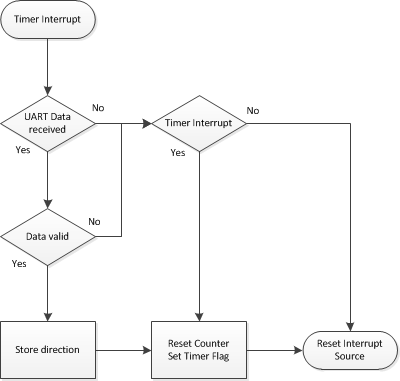
Nach Drücken der Zahl wird noch etwa eine Sekunde gewartet bevor man automatisch zum Hauptmenü zurückkehrt.

# Microcontroller Code

Die Software auf dem Carme Kit läuft grösstenteils in C und bedient sich der uC/GUI Library für die graphische Darstellung auf dem LCD. Das einzige Modul in Assembler ist der Interrupt-Handler und die dazu notwendigen Initialisierungen für den Timer und die UART0-Schnittstelle. Die Software kontrolliert das Grafik-Display, empfängt die Pfeiltasten vom PC und steuert den Spielablauf (streut Futter, kontrolliert die Schlangenlänge, prüft auf Schlangen-Wandkollision, Schlangen-Schlangenkollision und Schlange-Futterkollision).

## Interrupt Handler

Der Interrupt Handler wurde vor allem von Marcel Bärtschi entwickelt. Der Interrupt Handler dient zur Erkennung der Interruptquelle und anschliessendem Entscheiden, was ausgeführt werden soll. In der jetzigen Version ist als Interruptquelle nur der Timer0 benutzt. Geplant war zu beginn, die Datenübertragung auch mit Interrupts zu lösen, jedoch hatten wir etwas Probleme mit der Initialisierung der UART-Interrupts. So entschieden wir uns, die UART Daten zu pollen. Der Code, der den UART Interrupt abfangen würde, ist bereits (auskommentiert) vorhanden. Nun wird bei jedem Interrupt der ausgelöst wird zuerst abgefragt, ob Daten empfangen wurden, und erst danach wird die eigentliche Interrupquelle behandelt. Da wir nur den Timerinterrupt eingeschaltet haben, gibt es nun jede Millisekunde einen Interrupt.

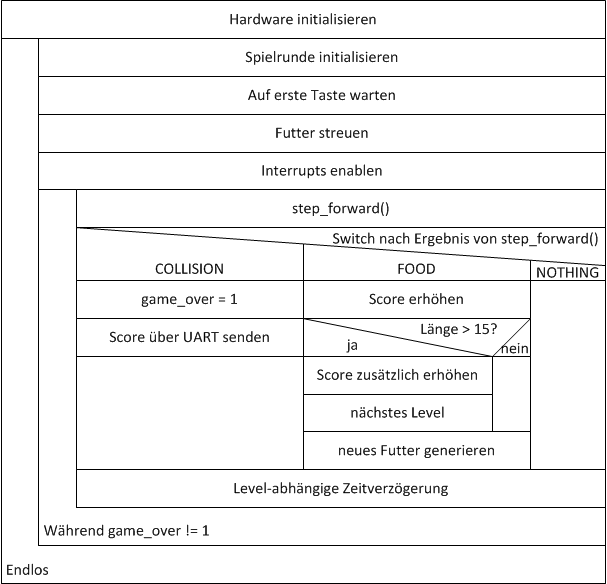


Wenn Daten angekommen sind, wird geprüft, ob die Daten gültig sind. Diese Prüfung ist eigentlich überflüssig, da vom Client nur gültige Daten gesendet werden. Jedoch können so allfällige Fehler der Übertragung herausgefiltert werden und das Programm funktioniert auch, wenn man irgendein ungültiges Zeichen aus dem Hyperterminal (oder äquivalentem Programm) schickt.

Nach der Prüfung wird die Richtung in einer globalen Variable abgespeichert und anschliessend der Timer Interrupt behandelt. Dort wird ein Timer Flag gesetzt, welches in der Delay Funktion als Zeitreferenz dient und anschliessend der Counter zurückgesetzt. Danach werden alle behandelten Interruptquellen zurückgesetzt und wieder ins „normale“ Programm zurückgekehrt.

## main

Die Funktionen im main-Modul wurden von Marcel Bärtschi und Cyril Stoller geschrieben. Hier ist vor allem die Main-Funktion zu erwähnen. Sie steuert den logischen Spielablauf.



Zuerst werden alle nötigen Initialisierungen vorgenommen und eine neue Spielrunde gestartet. Die Schlange bleibt jetzt solange als Punkt in der Mitte stehen, bis eine Pfeiltaste ans Carme Kit gesendet wird. Mit dieser Pfeiltaste wird nun das erste Futter gestreut und die Schlange beginnt sich in die entsprechende Richtung zu „entrollen“.

Dabei ist der Ablauf immer derselbe: Die Schlange führt einen Schritt in die momentan aktuelle Richtung aus.

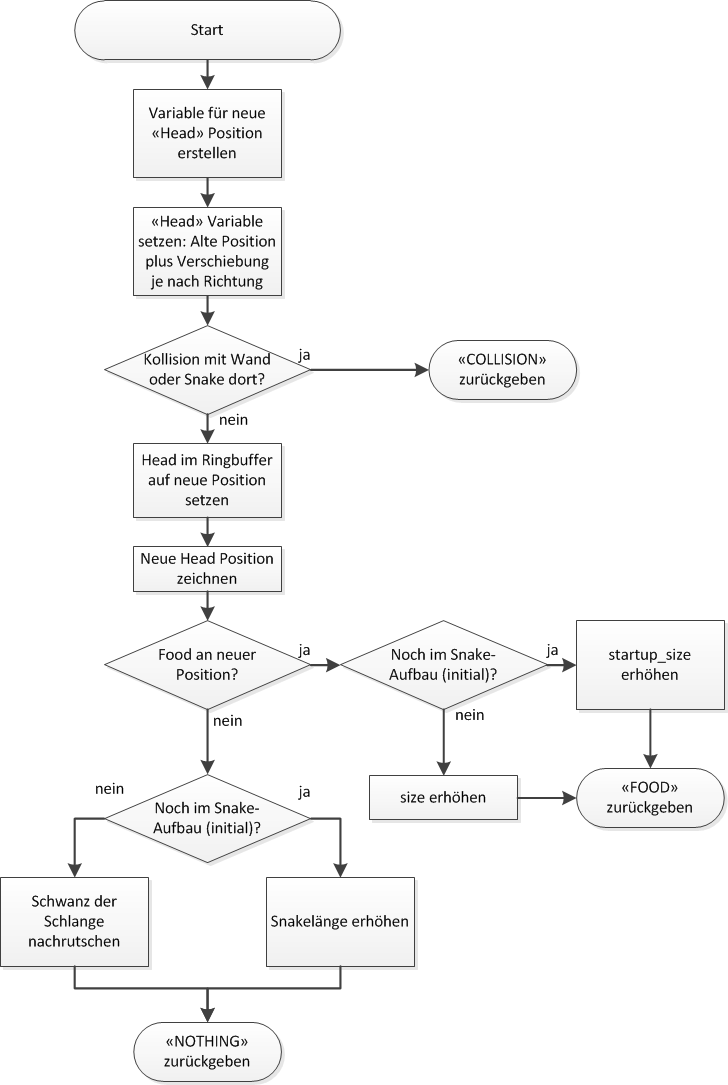
* Lautete das Resultat der Funktion step\_forward *COLLISION*, wird nun die Punktzahl über UART0 an den PC gesendet und das Spiel beendet.
* Lautete das Resultat *FOOD*, wird die Punktzahl erhöht. Dann wird geprüft, ob bei genügend langer Schlange ein neuer Level gestartet wird und schlussendlich wird noch neues Futter gestreut.
  + Wird ein neues Level gestartet, wird die Schlange wieder auf ihre ursprüngliche Grösse reduziert, der Punktestand um 10 zusätzliche Punkte erhöht und die Spielgeschwindigkeit erhöht (also die Zeitverzögerung zwischen den Schlangenschritten verkleinert).
* Lautete das Resultat NOTHING, passiert nichts und die Schlange fährt weiter.

Nach dieser Auswertung wird nun noch eine gewisse Zeit gewartet. Für diese Zeitverzögerungsfunktion wird der Timer eingesetzt, der jede Millisekunde ein Interrupt auslöst. Damit kann dann die Funktion delay\_time eine variable Anzahl Millisekunden verzögern.

Wird das Spiel beendet, beginnt der Zyklus wieder von Vorne: Der Bildschirm wird bis auf den Startpunkt der Schlange gelöscht und es wird auf eine eine erste Pfeiltaste gewartet.

## snake\_controller

Die Funktionen im Modul snake\_controller wurden von Cyril Stoller geschriebenen. Die hier wichtigste Funktion ist die Funktion step\_forward. Sie ist für das Verhalten der Schlange verantwortlich.



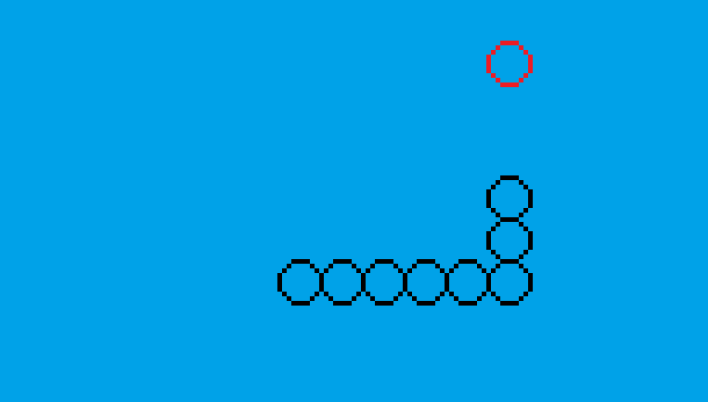
Am Anfang wird die neue Position bestimmt, welche sich durch die alte Position und die momentane Laufrichtung ergibt. Dann wird dieses „Zielfeld“ überprüft. Ist es ausserhalb des Spielfeldes, wird *COLLISION* zurückgegeben.

Wenn nicht, kann die Schlange dort gezeichnet werden. Nun ist noch herauszufinden, ob die Schlange dort Futter gefressen hat. Wenn ja, wird die Schlange verlängert (bei einer andern Variable, je nachdem, ob sie noch im Aufbau ist) und *FOOD* zurückgegeben.

Wenn dies auch nicht der Fall war, trat die Schlange auf ein leeres Feld. Nun wird noch geprüft, ob sie im Aufbau ist, d.h. ob sie noch immer automatisch wächst. Je nachdem wird der Schwanz gelöscht oder nicht. Am Schluss wird noch *NOTHING* zurückgegeben.

## Graphics

Die Grafik-Funktionen im Modul graphics wurden von Marcel Bärtschi geschrieben. Sie beherbergt Interfacefunktionen um die Zeichnungsfunktionen der uC/GUI Library mit den Spielfeldnormierten Positionsangaben nutzen zu können. Die Grafik haben wir sehr einfach gehalten. Wir zeichnen die ganze Schlange und das Futter nur als Kreisringe. Diese brauchen wenig Zeit um auf dem LC-Display angezeigt zu werden.



Zu Beginn des Spiels wird ein blauer Hintergrund gezeichnet, worauf sich die Schlange danach bewegt. Um möglichst viel Zeit zu sparen, werden immer nur die Daten auf dem Display gezeichnet, die verändert wurden. Mit der uC/GUI Library ist es sehr einfach, Flächen, Kreise und andere Formen zu zeichnen. Daher beschränken sich die Grafikfunktionen darauf, die Farbe des jeweils zu zeichnenden Objekts zu wählen und danach die entsprechende Funktion aus der Library aufzurufen.

Die Schlange besteht zu Beginn aus vier Kreisen, wobei ein Kreis einen Durchmesser von zehn Pixeln hat. Dies entspricht auch gleich dem Raster, in welches wir den Bildschirm aufgeteilt haben. Dass will sagen, man kann nur in Bahnen im Abstand von zehn Pixeln fahren.

# Spielbeschrieb

Im Spiel „Carme-Snake“ geht es darum, möglichst viel Futter aufzunehmen, ohne dabei in eine Wand oder sich selbst hineinzufahren. Dabei wird die Schlange bei jedem Happen ein Glied länger, was zu einer immer enger werdenden Situation führt. Um zusätzlich eine Schwierigkeit einzubauen, wird das Spiel schneller, sobald 15 Futterringe gefressen wurden. Das Erreichen des nächsten Levels hat auch einen Punktebonus zur Folge. Wenn man schlussendlich doch kollidiert, muss wieder bei Level 1 begonnen werden.

# Bedienungsanleitung

Um das Spiel überhaupt zum Laufen zu bringen, muss ein Carme Kit und die Carme DIE vorhanden sein. Weitere Informationen dazu unter: [www.carme.bfh.ch](http://www.carme.bfh.ch)

Zudem muss das Carme Kit mit einem Computer verbunden werden. Einerseits über USB um das Spiel auf das Kit zu laden, und andererseits über RS232 um die Schlange über den Computer zu steuern.

Wenn alles verbunden ist und der Code erfolgreich heruntergeladen, muss auf dem Computer die Client-Software „Carme-Snake.exe“ ausgeführt werden.

Zuerst erscheint das Hauptmenü. Um das Spiel zu starten, muss die Taste 1 gedrückt werden, anschliessend den Namen eingeben und mit Enter bestätigen. Spätestens jetzt muss das Carme-Kit mit einem RS232-Kabel mit dem Computer verbunden und die Software darauf gestartet sein. Um nun die Schlange zu starten, kann in eine beliebige Richtung gedrückt werden. Das Spiel beginnt beim Drücken der ersten Taste. Sobald das Spiel vorbei ist, wird dies angezeigt und man kann mit einem beliebigen Tastendruck wieder in das Hauptmenü zurückkehren.

Den Comport, auf dem das Carme-Kit verbunden ist, kann man im Hauptmenü mit der Taste 3 ändern. Man gelangt dadurch auf einen Bildschirm, der einen auffordert, die Comport-Nummer einzugeben. Nach drücken der Zahl kehrt man wieder in das Hauptmenü zurück.

Mit der Taste 2 kann eine Highscore-Liste angeschaut werden, die nach jedem Spiel aktualisiert wird (Das File „highscore.txt“ muss im dazu im Ordner der ausführbaren Datei vorhanden sein).

# Mögliche Erweiterungen

Da während der Projektarbeit die Zeit beschränkt war, haben wir gewisse Features weggelassen, und schon in der Zielsetzung einen gewissen Rahmen definiert. Zudem kamen uns im Laufe des Projekts noch Ideen, welche das Spiel noch interessanter gestalten oder den Code optimieren würden.

* UART Übertragung als Interrupt (Code auskommentiert vorhanden, aber nicht funktionsfähig)
* Eingabebuffer erstellen (für schnelle Tastendrücke)
* Grafik verschönern

# Fazit

Das Projekt war ein guter Ausgleich zum Unterricht und wir konnten viel Stoff der Theorie in der Praxis anwenden. Es war auch hilfreich, dass wir einen grossen Teil in C schreiben konnten. Dies wiederspiegelt ein Projekt, wie man es in der Praxis auch antreffen könnte. Wir konnten so auch gerade die Schnittstelle C zu Assembler und umgekehrt etwas besser begreifen. Schwierigkeiten hatten wir vor allem beim Entwickeln der Assemblerfunktionen.

# Source Code Hosting

Das gesamte Projekt ist unter GitHub online verfügbar:

Carme-Kit Code Repository: <https://github.com/stocyr/Carme-Snake>  
Client Software Code Repository: <https://github.com/stocyr/Carme-Snake-Client>

Dort kann auch überprüft werden, wer wann welches Sourcefile wo bearbeitet hat.

# Weiterführende Dokumentation mit Doxygen

Der gesamte Code, der von uns geschrieben wurde (also exklusiv der Marsenne Twister Bibliothek) ist mit Doxygen Kommentaren Dokumentiert. Die Dokumentation dazu kann als weiterführendes und vor allem sehr Code-spezifisches Nachschlagewerk dienen. Sie wurde als HTML-Dokumentation bereits kompiliert und liegt dem Projekt bei.