Carme-Snake

Inhalt

[Übersicht 2](#_Toc345927984)

[Computer 2](#_Toc345927985)

[Carme Kit 2](#_Toc345927986)

[Datenstrukturen 2](#_Toc345927987)

[Enumerations 2](#_Toc345927988)

[direction 2](#_Toc345927989)

[result 2](#_Toc345927990)

[Strutures 3](#_Toc345927991)

[location 3](#_Toc345927992)

[Globale Variablen im Main-Modul 3](#_Toc345927993)

[snake\_direction (volatile location) 3](#_Toc345927994)

[timer\_irq\_flag (volatile int) 3](#_Toc345927995)

[food (pos) 3](#_Toc345927996)

[level (int) 3](#_Toc345927997)

[Modulaufteilung 3](#_Toc345927998)

[PC client 4](#_Toc345927999)

[Interrupt Handler 4](#_Toc345928000)

[Main 4](#_Toc345928001)

[snake\_controller 4](#_Toc345928002)

[Graphics 5](#_Toc345928003)

[Mögliche Erweiterungen 6](#_Toc345928004)

[Fazit 6](#_Toc345928005)

# Installations und Bedienungsanleitung

Das gesamte Projekt ist unter <https://github.com/stocyr/Carme-Snake>

# Übersicht



Der Computer ist für das einlesen der Pfeiltasten zuständig. Er sendet diese Befehle über ein eigen dafür entwickeltes Protokoll über EIA232 an das Carme Kit.

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

## Computer

Die Software auf dem PC läuft in C mit der GUI Library von Ivo Oesch. Zuerst erscheint ein GUI, worin das Spiel gestartet bzw. beendet werden kann. Mit einer zusätzlichen Option kann eine Highscore Liste angeschaut werden, die nach jedem Spiel aktualisiert wird (Das File „highscore.txt“ muss im Ordner Release vorhanden sein).

Um das Spiel zu starten, muss die Taste 1 gedrückt werden, anschliessend den Namen eingeben und mit Enter bestätigen. Um nun die Schlange zu starten, kann in eine beliebige Richtung gedrückt werden. Die Schlange bleibt solange in der Mitte stehen, bis ein Zeichen ans Carme Kit gesendet wird. 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*.

## Carme Kit

Die Software auf dem Carme Kit läuft grösstenteils in C. Das einzige Modul in Assembler ist der Interrupt Handler und die dazu notwendigen Initialisierungen für den Timer und die UART-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).

# Datenstrukturen

## Enumerations

### direction

right = ‘r’, up = ‘u’, left = ‘l’, down = ‘d’

Diese Struktur wird benutzt, um die ankommenden Zeichen den Richtungen zuzuordnen.

### result

NOTHING, FOOD oder COLLISION

Diese Struktur wird von der Funktion step\_forward() benutzt, um der main() Funktion mitzuteilen, was beim Vorwärtsgehen geschehen ist. Entweder nichts, die Schlange ist auf Futter gestossen oder die Schlange ist kollidiert.

## Strutures

### location

Diese Struktur enthält einen X- und einen Y-Wert, der eine Position repräsentiert. Sie kann entweder in Pixel oder in Rasterpunkten (also Spielbrettposition) angegeben sein.

## Globale Variablen im Main-Modul

### snake\_direction (volatile location)

Globale Variable, welche vom Interrupthandler über UART aktualisiert wird.

Die Variable ist das Interface zur UART Schnittstelle. Sie enthält die Richtung, in die die Schlange steuern soll. Wird per PC Tastatur gesteuert und dann per UART zum CARME Kit gesendet.

Bevor überhaupt eine Taste gedrückt wurde hat die Variable den Wert '?', damit man erkennen kann, wenn dann erstmals ein r, o, l oder u hereinkommt.

### timer\_irq\_flag (volatile int)

Globale Variable, welche vom Interrupthandler über den Timer gesetzt wird.

Alle 1ms wird das Flag vom Interrupt des Timers gesetzt. Damit werden genaue Zeitverzögerungen möglich.

### food (pos)

Ist die momentane Position des Futters.

### level (int)

Ist der momentane Level des Spiels.

#### snake (Array aus location)

Ist ein Ringbuffer, welcher die Schlage repräsentiert. Dazu gehören folgende Variablen:

#### head (int)

#### tail (int)

#### size (int)

# Modulaufteilung

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

Ein weiteres Modul ist Assembler geschrieben und behandelt alle Hardwarenahen Funtionen. Es empfängt Pfeiltasten und aktualisiert die entsprechende Richtung in einer globalen Variablen (definiert in einer C-Datei). Ausserdem kümmert es sich um die Timer-Interrupts, welche für die Zeitverzögerung gebraucht werden.

Dann gibt es noch drei Module, welche sich um das eigentliche Spiel kümmern: main, graphics und snake\_controller. Es gibt ebenfalls noch das Modul marsenne, welches eine Bibliothek enthält um Pseudozufallszahlen zu generieren. Es wurde wie es ist 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

Der PC Client wurde von Peter Ambühl geschrieben.

## Interrupt Handler

Der Interrupt Handler wurde vor allem von Marcel Bärtschi entwickelt.

## Main

Die Funktionen im main-Modul wurden von Marcel Bärtschi und Cyril Stoller geschrieben.

|  |  |  |
| --- | --- | --- |
| **randomize\_location** | | **main.c** |
| **Beschreibung** | Liefert innerhalb der Spielfeldgrösse eine zufällige Position zurück | |
| **Input** | - | |
| **Output** | Zufallsposition | |

|  |  |  |
| --- | --- | --- |
| **Randomize\_food** | | **main.c** |
| **Beschreibung** | Liefert innerhalb der Spielfeldgrösse eine zufällige Position zurück, welche NICHT auf der Schlange liegt | |
| **Input** | - | |
| **Output** | Zufallsposition | |

|  |  |  |
| --- | --- | --- |
| **delay** | | **game\_logic.c** |
| **Beschreibung** | delays the processing by a certain milliseconds | |
| **Input** | delay time (int) | |
| **Output** | - | |

|  |  |  |
| --- | --- | --- |
| **restore\_interruptstate** | | **game\_logic.c** |
| **Beschreibung** | Stellt einen gespeicherten Interruptzustand wieder her | |
| **Input** | Gespeicherter Interruptstatus | |
| **Output** | - | |

|  |  |  |
| --- | --- | --- |
| **main** | | **main.c** |
| **Beschreibung** | main function | |
| **Input** | - | |
| **Output** | returns 0 | |

## snake\_controller

Die Funktionen im Modul snake\_controller wurden von Cyril Stoller geschribenen.

|  |  |  |
| --- | --- | --- |
| **step\_forward** | | **snake\_controller.c** |
| **Beschreibung** | steps the snake one step forward. tells the executing function, what has happened while stepping forward: Enum “result” | |
| **Input** | - | |
| **Output** | result of stepping forward (enum result) | |

|  |  |  |
| --- | --- | --- |
| **check\_wall\_collision** | | **snake\_controller.c** |
| **Beschreibung** | checks if the snake collided with the wall | |
| **Input** | position (location) | |
| **Output** | true / false | |

|  |  |  |
| --- | --- | --- |
| **check\_snake\_collision** | | **snake\_controller.c** |
| **Beschreibung** | checks if the snake collided with itself | |
| **Input** | position (location) | |
| **Output** | true / false | |

|  |  |  |
| --- | --- | --- |
| **check\_food\_collision** | | **snake\_controller.c** |
| **Beschreibung** | checks if the snake collided with a food piece | |
| **Input** | position (location) | |
| **Output** | true / false | |

|  |  |  |
| --- | --- | --- |
| **check\_initial\_state** | | **snake\_controller.c** |
| **Beschreibung** | Überprüft, ob die Schlange sich noch im Aufstartmodus befindet | |
| **Input** | - | |
| **Output** | true / false | |

|  |  |  |
| --- | --- | --- |
| **init \_snake** | | **snake\_controller.c** |
| **Beschreibung** | Initialisiert den Ringbuffer der Schlange. | |
| **Input** | - | |
| **Output** | - | |

## Graphics

Die Grafik-Funktionen im Modul graphics wurden von Marcel Bärtschi geschrieben.

|  |  |  |
| --- | --- | --- |
| **init\_graphics** | | **graphics.c** |
| **Beschreibung** | Initialisiert die GUI Funktionen der mitgelieferten CARME GUI Library | |
| **Input** | - | |
| **Output** | - | |

|  |  |  |
| --- | --- | --- |
| **draw\_snake\_head** | | **graphics.c** |
| **Beschreibung** | draws a new snake piece on the head of the snake | |
| **Input** | position (location) | |
| **Output** | - | |

|  |  |  |
| --- | --- | --- |
| **clear\_snake\_tail** | | **graphics.c** |
| **Beschreibung** | clears the snakes tail at the end of the snake | |
| **Input** | position (location) | |
| **Output** | - | |

|  |  |  |
| --- | --- | --- |
| **draw\_field** | | **graphics.c** |
| **Beschreibung** | draws a rectangle representing the playing field | |
| **Input** | - | |
| **Output** | - | |

|  |  |  |
| --- | --- | --- |
| **draw\_food** | | **graphics.c** |
| **Beschreibung** | draws a new food piece | |
| **Input** | Food (location) | |
| **Output** | - | |

# 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
* Auswahl des Serialport im GUI

# 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.