**Projektarbeit Informatik**

Softwaredokumentation   
LaserChess V1.2



Website: [stocyr.github.com/LaserChess](http://stocyr.github.com/LaserChess/)

Dozent: Ivo Oesch

Autoren: Marcel Bärtschi, Jascha Haldemann, Nicola Käser, Cyril Stoller

2 Semester 2012

**Inhaltsverzeichnis**

[1 Einleitung 1-3](#_Toc328130605)

[2 Richtlinien 2-4](#_Toc328130606)

[3 Analyse 3-6](#_Toc328130607)

[3.1 Pflichtenheft 3-6](#_Toc328130608)

[4 Design 4-7](#_Toc328130609)

[4.1 Grobdesign 4-7](#_Toc328130610)

[4.2 Detaildesign 4-8](#_Toc328130611)

[4.3 Datenstrukturen 4-8](#_Toc328130612)

[4.3.1 Enumerations 4-8](#_Toc328130613)

[4.3.2 Strukturen 4-10](#_Toc328130614)

[4.3.3 Globale Variablen 4-11](#_Toc328130615)

[4.3.4 Sonstige Datensätze 4-11](#_Toc328130616)

[4.3.5 Makros 4-13](#_Toc328130617)

[4.3.6 Definitionen 4-13](#_Toc328130618)

[4.4 Modulbeschreibungen 4-15](#_Toc328130619)

[4.4.1 Mainfunktionen (Laserchess) 4-15](#_Toc328130620)

[4.4.2 Spielfunktionen 4-17](#_Toc328130621)

[4.4.3 Logikfunktionen 4-18](#_Toc328130622)

[4.4.4 Grafikfunktionen 4-20](#_Toc328130623)

[4.5 Struktogramme und Flussdiagramme 4-24](#_Toc328130624)

[4.5.1 Modul LaserChess.c 4-24](#_Toc328130625)

[4.5.2 Modul Spiel.c 4-28](#_Toc328130626)

[4.5.3 Modul Logik.c 4-30](#_Toc328130627)

[4.5.4 Modul Grafik.c 4-35](#_Toc328130628)

[5 Implementation 5-38](#_Toc328130629)

[6 Zukünftige Erweiterungen 6-39](#_Toc328130630)

[6.1 Spiellogik 6-39](#_Toc328130631)

[6.1.1 Splitter 6-39](#_Toc328130632)

[6.1.2 Dynamische Startparameter 6-39](#_Toc328130633)

[6.1.3 Game over LAN 6-41](#_Toc328130634)

[6.1.4 Game-AI 6-41](#_Toc328130635)

[6.1.5 Snake-Walls 6-41](#_Toc328130636)

[6.2 Grafik 6-42](#_Toc328130637)

[6.2.1 Gif Animationen 6-42](#_Toc328130638)

[6.2.2 Drag-and-Drop 6-42](#_Toc328130639)

[6.2.3 Lasergeschwindigkeit 6-42](#_Toc328130640)

[6.2.4 Grafikoptionen Real-Time 6-42](#_Toc328130641)

[7 Anhang 7-43](#_Toc328130642)

# 

# Einleitung

Dieses Dokument dokumentiert das Softwareprojekt „LaserChess“. Die genaue Beschreibung der Aufgabenstellung ist im Dokument „Projektdokumentation“ enhalten. Das Pflichtenheft, welches im Voraus für dieses Projekt erstellt wurde und alle Grundideen enthält, ist im Anhang enthalten.

# Richtlinien

Für dieses Projekt gelten folgende Richtlinien:

Jedes Modul besteht grundsätzlich aus zwei Dateien, einer Schnittstellendefinitionsdatei (Headerdatei .h-File) und einer Implementationsdatei (.c-File). Die **main()**-Fuktion ist in der Haupt-C-Datei LaserChess.C enthalten. Alle projektweite Typendefinitionen sind in der Datei LaserChess.H enthalten. Die Namen der Header und der Implementationsdatei entsprechen jeweils dem Modulnamen, soweit möglich und sinnvoll.

Der Selbstschutz in den Headerdateien erfolgt in der Form

**#ifndef NAME**

**#define NAME**

**<Schnittstellendefinition>**

**#endif**

Wobei NAME wie folgt aufgebaut ist **DATEINAME\_H**. Jedes Modul ist mit einem Modulheader ausgestattet, jeweils in der Schnittstellen und der Implementationsdatei. Der Header enthält den Modulnamen, den Dateinamen, einen kurzen Funktionsbeschrieb zum Modul, die Namen aller zur Verfügung gestellten Funktionen und den Namen des ursprünglichen Autors. Die History jedes Files kann auf [github.com/stocyr/LaserChess/tree/master/src](https://github.com/stocyr/LaserChess/tree/master/src) angesehen werden. Zuerst klickt man auf den Namen des gewünschten Files, dann auf den Button **Blame** oben rechts:



Jede Funktion ist mit einem Funktionsheader ausgestattet. Der Header enthält den Funktionsnamen, einen kurzen Funktionsbeschrieb, die Namen und Funktion der Argumente, eine Beschreibung des Rückgabewertes und den Namen des Autors. Funktionen sollen klar definierte Aufgaben haben. Namen von Bezeichnern sollen Aussagekräftig sein, und die Funktion des entsprechenden Objektes erklären oder andeuten.

Modul- und Funktionsheader werden in englischer Sprache geschrieben. Kommentare im Code drin jedoch sind auf Deutsch verfasst und immer von der Art **// <Kommentar>** damit ganze Codeblöcke zu debug-Zwecken mit **/\* ... \*/** auskommentiert werden können. Der Code ist grundsätzlich mit sinnvollem und aussagekräftigem Kommentar zu versehen.

Makros (**#define**) werden grundsätzlich in Grossbuchstaben geschrieben, zu Strukturierung können Underscores verwendet werden. Beispiele: **PI**, **MAXIMAL\_FIELD\_WIDTH**

Enums werden grundsätzlich mit einem grossen Anfangsbuchstaben geschrieben. Structs werden grundsätzlich kleingeschrieben. Variablen werden grundsätzlich kleingeschriebenen und nötigenfalls mit Underscore "\_" in Wörter aufgeteilt. Funktionen werden grundsätzlich kleingeschriebenen und nötigenfalls mit Underscore "\_" in Wörter aufgeteilt.

Für jedes Modul ist grundsätzlich ein Programmierer verantwortlich. Dieser ist für die Sauberkeit und das Nachführen des Modulheaders zuständig. Andere Programmierer dürfen jedoch ebenfalls Änderungen in diesem Modul vornehmen.

# Analyse

## Pflichtenheft

Im Pflichtenheft, welches im Anhang vorliegt, wurde schon relativ viel Analysearbeit gemacht:

* die Regeln des Spiels wurden definiert,
* das Spielfeld festgelegt,
* die Spielfiguren inklusive einer möglichen Datenstruktur beschrieben,
* eine Menu-Führung definiert,
* die möglichen Spielzüge definiert,
* das Spielfeld als map-Datenbank definiert und
* Details der Laserfunktionalität vorgestellt.

# Design

## Grobdesign

Es werden vier verschiedene Module erstellt:

* Hauptmodul **LaserChess.c**
  + Das Modul LaserChess beinhaltet die Mainfunktionenen.
* Spielmodul **Spiel.c**
  + Das Spielmodul organisiert und kontrolliert das ganze Spiel.
* Logikmodul **Logik.c**
  + Das Logikmodul enthält vor allem die Logik des Laserabschusses. Ausserdem enthält sie verschiede Helper-Tools und Konvertierungsroutinen welche von den anderen Modulen genutzt werden.
* Grafikmodul **Grafik.c**
  + Es stellt der Software die Funktionen für die grafische Umsetzung zur Verfügung. Alles was grafisch ausgegeben wird, wird von diesem Modul erledigt.

Die Gründe für diese Aufteilung sind:

* Zu viele kleine Untermodule würden den Nachteil haben, dass extrem viele Querverbindungen zwischen den Modulen entstehen würden. Es gäbe viele Funktionen, die in einem ähnlichen „Aufgabenbereich“ arbeiten aber dennoch modular getrennt sind. Dies haben wir zu vermeiden versucht.
* Nur ein Modul in einem Softwareprojekt dieser Grössenordnung ist zu unübersichtlich.
* Bei dieser Aufteilung kann sehr bequem jeweils ein Teammitglied einen „Aufgabenbereich“ bzw. ein Modul übernehmen und sich darauf spezifizieren.

Grundsätzlich ist das Modul **LaserChess.c** zuoberst in der Hierarchie. Es arbeitet eng mit dem Modul **Spiel.c** zusammen. Diese beiden Module greifen dann vor allem auf die Logikfunktionen des Moduls **Logik.c** zu. Und dieses wiederum bedient sich für die grafische Umsetzung der „Befehle“ von oben der Funktionen im Modul **Grafik.c**.

## Detaildesign

Hier sieht man, wie die Funktionen der einzelnen Module miteinander zusammenarbeiten:



## Datenstrukturen

Hier werden die wichtigsten Datentypen und –Strukturen vorgestellt.

### Enumerations

#### Affiliation

PLAYER\_RED,iPLAYER\_BLUE  
Variablen dieses Typs, sind zur Bestimmung der Spieler. Sie werden gebraucht, um die Figuren dem entsprechenden Spieler zuzuordnen. Auch um zu speichern, welcher Spieler an der Reihe ist, wird eine Variable dieses Typs verwendet. Da die Members dieser Enumeration für die Zahlen 0 und 1 stehen, kann um den Spieler zu wechseln einfach die Variable invertiert werden.

#### Species

KING, MIRROR, SPLITTER, WALL, CANNON  
Enumeration für die verschiedenen Spielfiguren.

|  |  |
| --- | --- |
| KING: | Ist die Hauptfigur. Wenn der König getroffen wird, ist das Spiel zu Ende. Es spielt keine Rolle von welcher Seite er getroffen wurde. Der König kann auf dem Spielfeld bewegt werden, aber er kann nicht gedreht werden. |
| MIRROR: | Mit dem Spiegel kann der LASER-Strahl umgelenkt werden. Der Spiegel wird zerstört, wenn er an einer nicht reflektierenden Seitenwand getroffen wird. Er kann beliebig auf dem Spielfeld gedreht und verschoben werden. |
| SPLITTER: | Der Splitter ist eine optionale Erweiterung des Spiegels. Er teilt den LASER-Strahl in zwei Strahlen auf. Der Splitter kann nicht zerstört werden, ist frei dreh- und verschiebbar. |
| WALL: | Die Wand ist zum Schutz des Königs gedacht. Sie kann nicht zerstört und nicht gedreht werden. |
| CANNON: | Die Kanone feuert den LASER-Strahl ab. Sie kann verschoben und gedreht werden. |

#### Orientation

EAST, NORTH, WEST, SOUTH, NORTH\_EAST = 0, NORTH\_WEST = 1, SOUTH\_WEST = 2, SOUTH\_EAST = 3

|  |  |  |
| --- | --- | --- |
| KING, WALL, CANNON | Beschreibung: C:\Users\Marcel\Documents\Tech\Informatik\CarmeWorkspace\LaserChess\docs\img\direction.png | Für die Figuren King, Wall und Cannon verwenden wir für die Orientierung die vier Himmelsrichtungen, wobei Norden gegen oben zeigt. |
| MIRROR, SPLITTER | Beschreibung: C:\Users\Marcel\Documents\Tech\Informatik\CarmeWorkspace\LaserChess\docs\img\mirror_orientation.png | Da die Spiegel nicht gerade stehen, verwenden wir bei den Spiegeln die Zwischenrichtungen. Da eine Figur jeweils nur vier Richtungen annehmen kann, sind die Zwischenrichtungen auch von 0-3 in der Enumeration definiert. Damit vereinfachen wir die Handhabung. |

#### Direction

RIGHT, UP, LEFT, DOWN  
Richtungsvariable für die LASER-Funktion.

#### Angle

CCW = 1, CW = -1

Hilfsvariable für die Drehrichtung bei einem Winkel. 1 entspricht einer Drehung im gegen Uhrzeigersinn,  
-1 im Uhrzeigersinn.

#### Spielmodus

NORMALMODE, SETMODE, STARTOPEN, OPEN, EXIT, INVALID\_INPUT (EASTER\_EGG1, EASTER\_EGG2, EASTER\_EGG3)

Das Menü liefert einen der Spielmodi zurück.

* *Normalmode*: Lädt die vordefinierte Aufstellung
* *Setmode*: Die Spieler platzieren abwechslungsweise die Spielfiguren auf dem Spielfeld.
* *Open*: Man kann eine selbst definierte Aufstellung laden.
* *Exit*: Verlässt das Spiel.
* *Invalid\_Input*: Führt das Menü nochmals aus

#### Gamecontrol

EXIT\_GAME, KING\_RED\_DEST, KING\_BLUE\_DEST, NONE, MIRROR\_RED\_DEST, MIRROR\_BLUE\_DIST   
Die LASER-Funktion gibt zurück, was getroffen wurde, das wird in die Gamecontrol Variable gespeichert. Zusätzlich kann in der Variable gespeichert werden, ob das Spiel während der LASER-Funktion abgebrochen wurde.

#### Sound

LASER, REFLECTION, DESTRUCTION, VICTORY, IGNORE, INTRO, MUSIC, PLING, BELL

Sound-Effekte, welche auf die entsprechenden Audio-Files verweisen.

### Strukturen

#### location

Die Struktur enthält X- und Y-Koordinate als ganzzahlige Werte. Variablen dieses Typs werden sowohl für Pixelkoordinaten als auch für Mapkoordinaten benutzt.

#### pawn

Pawn ist ein Typedef einer Struktur für die Spielfiguren. Die Struktur enthält folgende Informationen:

* *PLAYER*  Enum Affiliation
* *TYPE* Enum Species
* *DIR* Enum Orintation
* *[Pos]* location [X,Y]

### **Globale** Variablen

#### AppPath

Eine Variable die den aktuellen Pfad der Applikation enthält. Wird benötigt um die Bilder zu laden.

#### map

Enthält die aktuelle Spielfeldsituation. Es ist ein zweidimensionales Array, standardmässig 8 x 6 Felder gross (mit Makros verstellbar), welches Pointer auf die Spielfiguren enthält. Wo keine Figur steht, ist ein NULL-Pointer.

#### Sound\_On

Speichert ob die Musik ein-, oder ausgeschaltet ist.

#### MapPath

Pfad der Map, falls direkt eine Map geöffnet wurde.

### Sonstige Datensätze

#### TITLE

Ascii-Art-Schriftzug für „LaserChess“. Wird bei der Begrüssung in der Konsole ausgegeben.

#### figure

Alle Figuren, welche als Pointer in der Map referenziert werden, sind auch als ‚richtige‘ Variablen vorhanden. Die Figuren sind in einem Array vom Typ Pawn gespeichert. Beim Initialisieren des Figure-Arrays wird auch gleich die Standardaufstellung geladen.

#### Standardaufstellung

Die vordefinierte Aufstellung der Figuren auf der Map sieht wie folgt aus:



#### Map-File

Um den Spielstand abzuspeichern oder zu laden, werden die Daten in Map-Files gespeichert. Alle Werte der Pawn-Struktur werden untereinander als Zahl in die Datei geschrieben. Die Reihenfolge wie die Figuren in der Datei gespeichert werden, ist analog zum Figure-Array. Für jede Figur fünf Zahlen gespeichert. Am Anfang des Files steht ein L, zur Identifizierung. Die Reihenfolge der Werte ist wie folgt:

L Kontroll-char  
0 PLAYER (0 = rot, 1 = blau)   
0 SPECIES (0 = KING, 1 = MIRROR, 2 = SPLITTER, 3 = WALL, 4 = CANNON)  
3 ORIENTATION (0 = EAST, 1 = NORTH, 2 = WEST, 3 = SOUTH)  
3 X-Position   
5 Y-Position

### Makros

#### Norm

Eine Hilfsmakrofunktion zum Drehen einer Figur. Um eine Figur zu drehen, muss die Richtungsvariable inkrementiert oder dekrementiert werden. Da die Enumeratoren für die Richtung nur von 0-3 gehen, werden Werte grösser 3 oder kleiner 0 wieder auf 0-3 normiert.

#### ROTATE\_LEFT

Dreht eine Figur nach links (CCW). Das heisst, die Richtungsvariable wird inkrementiert.

#### ROTATE\_RIGHT

Dreht eine Figur nach rechts (CW). Das heisst, die Richtungsvariable wird dekrementiert.

#### ABS

Liefert den Betrag eines Wertes zurück.

#### IS\_EVEN

Gibt 1 zurück, wenn der übergebene Wert Gerade ist, sonst 0.

#### RED\_FIG, BLUE\_FIG

Sind Hilfsmakros zum Adressieren der Figuren. Da die Figuren nach Farbe sortiert im Array gespeichert sind, aber die Spieler die Figuren abwechslungsweise setzen, teilt RED\_FIG den übergebenen Wert durch zwei. BLUE\_FIG teilt zuerst auch durch zwei, addiert danach noch Sieben. In einer nächsten Version könnte man das figure-Array zweidimensional machen, mit einer Unterscheidung der PLAYER.

### Definitionen

#### MAP\_DIR und MAP\_EXT

Ordnerpfad zu den vordefinierten Maps (Pfad relativ zu \*ApplicationPath) und Mapendung „.map“.

#### ANZ\_FIGURES

Definiert wie viele Figuren maximal vorhanden sind. Ist für die Standardaufstellung auf 14 eingestellt.

#### PLAYGROUND\_...\_MAX

Darin kann die Länge und die Breite des Spielfelds eingestellt werden. Standardmässig ist PLAYGROUND\_X\_MAX auf *acht* und PLAYGROUND\_Y\_MAX auf *sechs* eingestellt.

## Modulbeschreibungen

Die Modulbeschreibungen, die hier vor allem zwecks Interface-Dokumentation aufgelistet werden, sind genau so wie sie hier stehen auch als jeweilige Funktions-Header im Source-Code vorhanden.

### Mainfunktionen (Laserchess)

Das Modul LaserChess beinhaltet die Mainfunktionenen. Für das Modul war vor allem Marcel Bärtschi zuständig. Im Folgenden werden die Beschreibungen, Inputs und Outputs der Funktionen in diesem Modul aufgelistet:

|  |  |  |
| --- | --- | --- |
| **create\_figures** | | **LaserChess.c** |
| **Beschreibung** | Initializes all figures from a received figure array.  Sets figures to the default map-position.  Currently initializes 14 figures (Optional: Splitter not defined in this version). | |
| **Input** | Pointer to the original figure array in the main-procedure | |
| **Output** |  | |
| **menu** | | **LaserChess.c** |
| **Beschreibung** | User can choose between the modes: ‘NORMAL’, ‘SETMODE’ and ‘quit the game’. | |
| **Input** |  | |
| **Output** | Enum Mode | |

|  |  |  |
| --- | --- | --- |
| **set\_figure\_positions** | | **LaserChess.c** |
| **Beschreibung** | The player can set his pawn freely on the map.  The figures in the array are sorted by color.  To toggle the player: i/2 for red and (i/2)+7 for blue. | |
| **Input** | Pointer to Array of all the figures. | |
| **Output** | Returns -1 if exit button is pressed, otherwise 0. | |

|  |  |  |
| --- | --- | --- |
| **init\_game** | | **LaserChess.c** |
| **Beschreibung** | Initializes the game.  Calls the graphic-functions and places the figures. | |
| **Input** | Array of all the figures, play mode (to decide whether to place all the figures, to initialized state or let the users place them alternating). | |
| **Output** | If wild failure appears, returns 0, otherwise returns 1. | |

|  |  |  |
| --- | --- | --- |
| **clear\_map\_array** | | **LaserChess.c** |
| **Beschreibung** | Clears the map array (writes all positions to NULL). | |
| **Input** |  | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **argument\_handler** | | **LaserChess.c** |
| **Beschreibung** | Reads the start-arguments. If EXE was started by opening a map file, it tries to load and start a game.  If there are start-variables defined, they will be set.  Unknown arguments are printed to screen. | |
| **Input** | int argn, number of arguments; char\* args[], arguments; pawn \*figure,  Figure-array needed to start a game. | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **gfxmain** | | **LaserChess.c** |
| **Beschreibung** | Uber-main function. Will be called FIRST! | |
| **Input** | System console call parameters. (OS specific) | |
| **Output** |  | |

### Spielfunktionen

Das Spielmodul organisiert und kontrolliert das ganze Spiel. Für das Modul war vor allem Marcel Bärtschi zuständig. Im Folgenden werden die Beschreibungen, Inputs und Outputs der Funktionen in diesem Modul aufgelistet:

|  |  |  |
| --- | --- | --- |
| **create\_focus** | | **Spiel.c** |
| **Beschreibung** | Draws a green Background on all free Fields around the selected figure. | |
| **Input** | location struct (x-y-coordinates of selected figure) | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **clear\_focus** | | **Spiel.c** |
| **Beschreibung** | Draws an empty field to all marked fields. | |
| **Input** | location struct (x-y-coordinates of selected figure) | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **spiel** | | **Spiel.c** |
| **Beschreibung** | Handles the game: Treats the mouse inputs, execute laser(), displays winner, close graphics. | |
| **Input** | pawn \*figure (used for cannon-position) | |
| **Output** |  | |

### Logikfunktionen

Das Logikmodul enthält vor allem die Logik des Laserabschusses. Ausserdem enthält sie verschiede Helper-Tools und Konvertierungsroutinen welche von den anderen Modulen genutzt werden. Für das Modul war vor allem Cyril Stoller zuständig. Im Folgenden werden die Beschreibungen, Inputs und Outputs der Funktionen in diesem Modul aufgelistet:

|  |  |  |
| --- | --- | --- |
| **laser** | | **Logik.c** |
| **Beschreibung** | Draws the laser from the cannon across the whole playground and calls all the other functions handling figure behaviour. | |
| **Input** | Receives the field from which the laser shoot is done. This field is not painted with laser anymore, but the field NEXT to it, whose direction from the field is specified with dir. | |
| **Output** | If a wall or cannon was hit, or the laser passes out of the playground, return 0. If a king was hit: -1 for player\_red, -2 for player\_blue.  If a mirror was hit: +1 for player\_red, +2 for player\_blue.  (In case of a splitter being hit: then two laser paths are generated and the return value is the one with the higher priority (descending order): king, mirror, wall / cannon) | |

|  |  |  |
| --- | --- | --- |
| **is\_inside\_map** | | **Logik.c** |
| **Beschreibung** | Checks if the given coordinates are inside the array. | |
| **Input** | Given coordinates (map position) | |
| **Output** | If inside map (means, inside the range [0 - 7][0 - 5], then it returns 1.  Otherwise it returns 0 | |

|  |  |  |
| --- | --- | --- |
| **is\_figure** | | **Logik.c** |
| **Beschreibung** | Checks if the given coordinates (map position) contains a figure. | |
| **Input** | x and y as map position | |
| **Output** | If there is a figure, return 1.  If it’s an empty field, return 0  (A wall is threatened as a figure). | |

|  |  |  |
| --- | --- | --- |
| **move\_figure** | | **Logik.c** |
| **Beschreibung** | Moves a figure to the given location. | |
| **Input** | figure pointer, new playground location | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **destroy\_figure** | | **Logik.c** |
| **Beschreibung** | Destroys a figure (deletes it from the map array). | |
| **Input** | pawn \*figure | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **mouseclick\_to\_map** | | **Logik.c** |
| **Beschreibung** | Get Mouse-Clicks and returns the map position. | |
| **Input** |  | |
| **Output** | Returns location struct, of the field who was hit or ERROR when the click was beyond the map or there was no click. | |

|  |  |  |
| --- | --- | --- |
| **path\_handler** | | **Logik.c** |
| **Beschreibung** | Combines the two strings path and file after checking if there's enough memory available. | |
| **Input** | const char path[] - String with the path of file  char file[] - String with the filename | |
| **Output** | returns string with the complete path | |

|  |  |  |
| --- | --- | --- |
| **play\_sound** | | **Logik.c** |
| **Beschreibung** | Plays the sound of chosen enumeration. Switches all sounds off, if a sound was not found. | |
| **Input** | Enum: Laser, Reflexion, Destruction, Victory, Ignore, Intro, Music, Bling, Bell. | |
| **Output** |  | |

### Grafikfunktionen

Das Grafikmodul stellt der Software die Funktionen für die grafische Umsetzung zur Verfügung. Alles was grafisch ausgegeben wird, wird von diesem Modul erledigt. Für das Modul war vor allem Nicolas Käser und Jascha Haldemann zuständig.

Im Folgenden werden Beschreibungen, Inputs und Outputs der Funktionen in diesem Modul aufgelistet:

|  |  |  |
| --- | --- | --- |
| **DrawTransformedImage** | | **Grafik.c** |
| **Beschreibung** | Draws the given image scaled and rotated at the given position into the current image. | |
| **Input** | imageId: Handle of image to draw x, y position to draw image at…  scalex: Scalingfactor for x axis (float value)  scaley: Scalingfactor for y axis (float value)  Angle: Angle to rotate Image (in rad) | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_sharp\_empty\_rectangle** | | **Grafik.c** |
| **Beschreibung** | Draws empty rectangle with sharp edges. | |
| **Input** | x and y koord. as windowskoord.,  int Width, int Height, ColorType Color, int LineWidth | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **pixel\_to\_map** | | **Grafik.c** |
| **Beschreibung** | Convert windowskoord. to map position. | |
| **Input** | x and y as windowskoord. | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **map\_to\_pixel** | | **Grafik.c** |
| **Beschreibung** | Convert mappositon to windowskoord. (Point upper left) | |
| **Input** | x and y as map position | |
| **Output** |  | |
| **draw\_playground** | | **Grafik.c** |
| **Beschreibung** | Draws a playground (field and lines) | |
| **Input** |  | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **scale\_handler** | | **Grafik.c** |
| **Beschreibung** | Returns the percentage for scaling the image to field size. | |
| **Input** | image\_ID, a valid ID of a loaded image file | |
| **Output** | size scale,  x- and y-scale factor in percentage of the field size | |

|  |  |  |
| --- | --- | --- |
| **draw\_focus** | | **Grafik.c** |
| **Beschreibung** | Draws a focus at the selected field. | |
| **Input** | x and y as map position | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_rot\_focus** | | **Grafik.c** |
| **Beschreibung** | Draws a rotation-image on the selected field | |
| **Input** | x and y as map position | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_empty\_field** | | **Modul** |
| **Beschreibung** | Deletes the selected field (reset). | |
| **Input** | x and y as map position | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_half\_laser** | | **Grafik.c** |
| **Beschreibung** | Help-function for draw\_laser and draw\_angled\_laser.  Draws half of the laser in the selected field  (v1.1: Laser glows). | |
| **Input** | x and y as map position, direction | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_laser** | | **Grafik.c** |
| **Beschreibung** | Draws the laser in the selected field (2x draw\_half\_laser). | |
| **Input** | x and y as map position, direction | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_angled\_laser** | | **Grafik.c** |
| **Beschreibung** | Draws the angled laser in the selected field (2x draw\_half\_laser). | |
| **Input** | x and y as map position, direction, angle (-1 right, 1 left) | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **destroy\_images** | | **Grafik.c** |
| **Beschreibung** | Deletes with init\_images() loaded images from memory | |
| **Input** |  | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **init\_images** | | **Grafik.c** |
| **Beschreibung** | Loads images from files into memory | |
| **Input** |  | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_figure** | | **Grafik.c** |
| **Beschreibung** | Draws figure at its location with its rotation/direction | |
| **Input** | pawn \*figure (figurepointer) | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_figure\_destroyed** | | **Grafik.c** |
| **Beschreibung** | Draws/animates the destruction of a mirror.  (V1.0, it only draws an empty field  (V1.1, "Melting"-animation with rectangles  (V1.2, offset increases always 1 pixel, not laser width  (V1.3, New animation, with glow) | |
| **Input** | pawn \*figure | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_invert\_colors** | | **Grafik.c** |
| **Beschreibung** | Inverts the colours of the defined part. | |
| **Input** | x and y as windowskoord. for start position; width and height for the size | |
| **Output** |  | |

|  |  |  |
| --- | --- | --- |
| **draw\_winner\_text** | | **Grafik.c** |
| **Beschreibung** | Writes winner text on screen. | |
| **Input** | pawn \*hit\_king | |
| **Output** |  | |

## Struktogramme und Flussdiagramme

Hier werden die Struktogramme und Flow Charts aller Funktionen in den vier Modulen aufgelistet und die Funktion je nach Komplexität noch etwas beschrieben.

### Modul LaserChess.c

#### gfxmain()

Das ist die Hauptfunktion des Programms. Von hier aus werden alle anderen Funktionen aufgerufen. Die gfxmain-Funktion selbst wird von QT aufgerufen und ihr wird unter anderem der Applikationspfad übergeben, damit das Programm auf Dateien zugreifen kann.

Zuerst wird die Begrüssung in der Konsole ausgegeben und die Spielfiguren werden erstellt. Anschliessend läuft das Programm in einem Loop, solange bis das Spiel abgebrochen oder beendet wird. Vor jedem Durchlauf wird das Spielfeld, also das Pointer-Array, gelöscht, respektive mit Null-Pointern initialisiert. Dazu wird die Funktion **clar\_map\_array()** benutzt. Danach wird das Menü solange ausgeführt, bis eine gültige Eingabe getätigt wurde. Je nach Spielmodus, der gewählt wurde, wird nun das Spiel initialisiert und danach solange ausgeführt, bis ein Spieler gewonnen hat, oder das Spiel abgebrochen wird. Die Funktionen **init\_game()** und **spiel()** sind einzeln genauer beschrieben.



#### clear\_map\_array()



#### create\_figures()

C:\Users\Marcel\Documents\Tech\Informatik\CarmeWorkspace\LaserChess\docs\Struktogramme\LaserChess(main)\Init_figures.png

#### menu()



Damit der Spieler die verschiedenen Modi auswählen kann, ist das Menü vorhanden. Es wird von der Funktion **gfxmain()** ausgeführt, hat keinen Inputparameter und gibt den Modus als Enum zurück, der gewählt wurde. Da noch die Tokens für die Eastereggs abgefragt werden, wird mit gets() eingelesen, danach geschaut ob die erste Stelle eine Zahl ist und diese wird danach unterschieden. Das heisst, wenn zwei Zahlen eingegeben werden, wird nur die erste berücksichtigt. Entspricht die Eingabe nicht einem der Modi, wird standardmässig Ungültige Eingabe zurückgegeben.

#### init\_game()

Um das Spiel anzufangen, müssen zuerst noch einige Sachen eingestellt und initialisiert werden. Der Funktion werden ein Pointer auf das Figure-Array und der Spielmodus übergeben. Die Funktion liefert eins zurück, wenn erfolgreich initialisiert wurde, ansonsten Null.

Die Funktion lädt zuerst die Bilder. Wenn diese nicht vorhanden sind oder nicht geöffnet werden konnten, wird die Initialisierung abgebrochen und das Menü nochmals ausgeführt. Danach wird geschaut ob der Modus gewählt ist um die Figuren frei zu setzen oder eine externe Aufstellung geladen werden soll. Wenn nichts davon gewählt ist, wird die Standardaufstellung geladen. Um eine Externe Aufstellung zu Laden, muss ein File im Ordner des Spiels vorhanden sein, worin die Daten der Aufstellung gespeichert sind. Das File repräsentiert die Figuren, respektive das Figurearray. Alle Werte der Pawn-Struktur werden untereinander als Zahl gespeichert. So sind für jede Figur fünf Zahlen gespeichert. Am Anfang des Files steht ein L, zur Identifizierung. Die Reihenfolge der Werte ist wie folgt:

L Kontroll-char  
0 PLAYER (0 = rot, 1 = blau)   
0 SPECIES (0 = KING, 1 = MIRROR, 2 = SPLITTER, 3 = WALL, 4 = CANNON)  
3 ORIENTATION (0 = EAST, 1 = NORTH, 2 = WEST, 3 = SOUTH)  
3 X-Position   
5 Y-Position



#### set\_figure\_position()



Wenn die Spieler die Figuren selbst auf dem Spielfeld platzieren wollen, wird diese Funktion ausgeführt. Zuerst kann der erste Spieler irgendwo im Spielfeld eine Figur setzen. Er kann sie mit dem Mausrad solange rotieren, bis er ein zweites Mal auf das Spielfeld klickt, danach ist der andere Spieler am Zug. Dies wiederholt sich solange, bis alle Figuren auf dem Spielfeld gesetzt sind. Die Reihenfolge, in der die Figuren gesetzt werden ist durch die Reihenfolge vorgegeben, wie die Figuren im Figure-Array liegen. Zur Hilfe wird immer der Name der nächsten Figur in der Konsole ausgegeben.

### Modul Spiel.c

#### create\_focus()



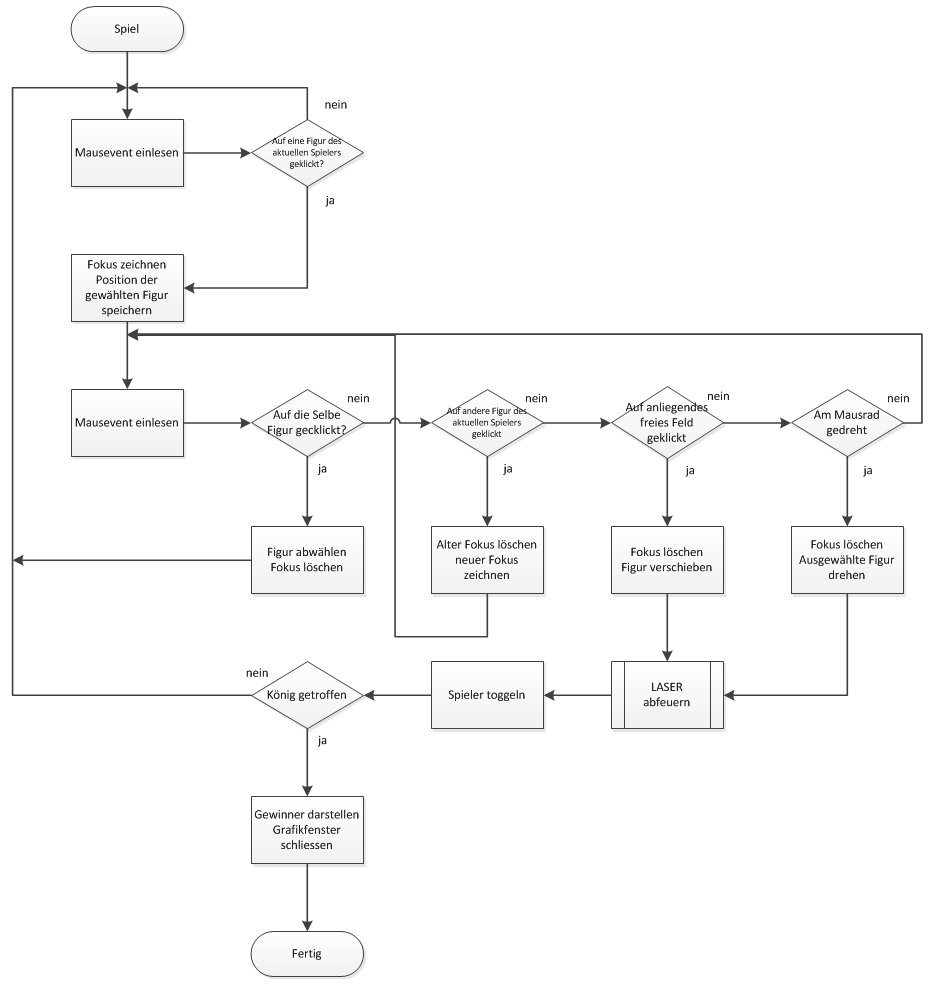
#### clear\_focus()



#### spiel()

Die Spielfunktion steuert den eigentlichen Spielvablauf. Übergeben wird der Funktion nur das Figurearray. Dies wäre eigentlich nicht nötig, da immer mit der Map gearbeitet wird. Jedoch benötigt die LASER-Funktion den Ort der Kanone und es ist einfacher dies aus dem Figurearray zu übergeben, als in der Map nach der aktuellen Kanone zu suchen. Auch um den Spielstand abzuspeichern wird einfach das Figurearray durchgegangen und alle Werte in die Datei geschrieben. Die Funktion hat keinen Rückgabewert. Wenn beendet wird, beginnt die Schlaufe in der Mainfunktion in jedem Fall von vorne.

Zuerst wird ein Mausereignis eingelesen und geschaut ob auf eine Figur des Spielers geklickt wurde, der im Moment am Zug ist. Wenn das der Fall ist, wird diese Figur selektiert. Danach wird wieder ein Mausereignis eingelesen und geschaut ob ein gültiger Zug ausgeführt wurde. Wenn das der Fall ist, wird der Zug ausgeführt und der Laser abgefeuert. Dies geht solange, bis einer der Könige getroffen wurde. Dann wird der Gewinner angezeigt, das Grafikfenster schliesst, und man kehrt ins Menü zurück.



### Modul Logik.c

#### laser()

Die Laserfunktion wird von der Funktion **spiel()** aufgerufen und bekommt als Argumente ein location-struct **pos** als map-Koordinate und ein Direction-Enum **dir** mit. Damit wird der Start des Laserabschusses festgelegt. **pos** ist das Feld, von dem aus der Laser abgeschossen wird und **dir** ist die Richtung, in der der Laser vom Feld aus gesehen abgeschossen wird. Diese Konvention erlaubt es, dass die Laserfunktion für jedes Feld neu aufgerufen wird, nur mit anderen Parametern.

Bei jeder Ausführung wird nun das Feld, in das der Laser „gehen“ wird, evaluiert und geprüft, auf was für eine Situation (Spiegel, König, WALL, Spielfeldbegrenzung oder einfach nur leeres Feld) der Laser stösst und was dann zu unternehmen ist. Am Schluss wird dann jeweils eine Funktion aus dem Grafikmodul ausgeführt (entweder **draw\_laser()** oder **draw\_laser\_angled()**). Diese Funktionen benötigen beide die Feldposition auf welcher gezeichnet werden muss (also wieder ein location-struct als map-Koordinate) und die Richtung, aus der der Laserstrahl durch dieses Feld hindurchwandert. Diese wird mit einem Direction-Enum weitergegeben. Zusätzlich benötigt die Funktion **draw\_laser\_angled()** noch den Parameter, in welche Richtung der abgewinkelte Strahl an einem Spiegel vorbeiwandert. Dieser wird mit einem Angle-Enum mitgegeben.

Nachdem der Laser jeweils auf Situationen gestossen ist, welche ihn nicht an der Fortbewegung gehindert haben (also kein zerstörter Spiegel oder König, keine WALL, keine Laserkanone und keine Spielfeldbegrenzung), wird also die Laserlinie gezeichnet. Dann führt er sich selbst wieder aus und gibt diesmal der **Laser()** Funktion als Startfeld das aktuell überstrichene Feld mit und als Richtungsangabe entweder dieselbe, die er schon vorher hatte (bei einem leeren Feld) oder eine um 90° CCW oder CW gedrehte Richtung (bei einer Umlenkung durch einen Spiegel).

Im Code wird nach dem rekursiven Aufruf der **Laser()** Funktion die Laserlinie gleich wieder gelöscht. Das bewirkt, dass die Funktion sich selbst zuerst solange rekursiv aufruft bis sie irgendwo anstösst oder etwas kaputt-schiesst und dann jeweils die Zeilen nach dem **Laser()** Funktionsaufruf ausgeführt werden. So wird nach einem beendeten Laserstrahl der gesamte Laserstrahl von den jeweiligen Funktionsaufrufinstanzen auf den Feldern, die der Laser überstrichen hat, selbstständig gelöscht.

Wichtig zu bemerken ist auch noch, dass die **Laser()** Funktion einen Rückgabewert hat. Dieser Rückgabewert wird jeweils von der aufrufenden Funktion zwischengespeichert und schlussendlich zurückgegeben, so dass der Rückgabewert alle Instanzen durchläuft und schlussendlich beim ersten Funktionsaufruf, bei der **spiel()** Funktion, landet.

Aktionen:

* Bei einer Spielfeldbegrenzung (also das nächste Feld ist *nicht mehr* im Spielfeld): keine Linie zeichnen, einfach beenden und einen entsprechenden Rückgabewert zurückgeben.
* Bei einem leeren Feld: gar nichts unternehmen, nur Laserlinie gerade zeichnen. Dann nach dem rekursiven Aufruf die Laserlinie wieder löschen.
* Bei einer WALL (welche den König schützt): keine Linie zeichnen, einfach beenden und einen entsprechenden Rückgabewert zurückgeben.
* Bei einem König: die Zerstörung des Königs per Grafik-Modul-Funktion zeichnen und einen entsprechenden Rückgabewert zurückgeben.
* Bei einem Spiegel: je nachdem ob der Spiegel an der schrägen Fläche getroffen wird, wo er reflektiert, wird die Reflektion gezeichnet und dann nach dem rekursiven Aufruf die Laserlinie wieder gelöscht. Ansonsten wird die Spiegelzerstörung per Grafik-Modul-Funktion gezeichnet und einen entsprechenden Rückgabewert zurückgeben.
* Bei einer Laserkanone: keine Linie zeichnen, einfach beenden und einen entsprechenden Rückgabewert zurückgeben.

Der Ablauf ist hier noch einmal grafisch mit einem Flow Chart dargestellt:



#### is\_inside\_map()



#### is\_figure()



#### move\_figure()



#### destroy\_figure()

G:\BFH\Struktogramme Logik.c\destroy_figure.png

#### mouseclick\_to\_map()



#### path\_handler()



#### map\_extension\_handler()



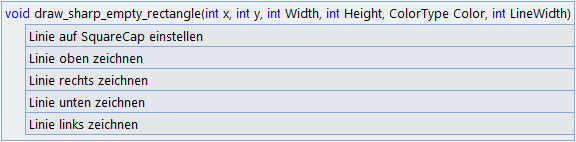
#### play\_sound()



### Modul Grafik.c

#### DrawTransformedImage

#### draw\_sharp\_empty\_rectangle

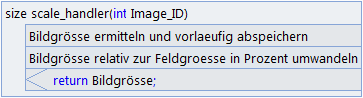


#### pixel\_to\_map

#### map\_to\_pixel

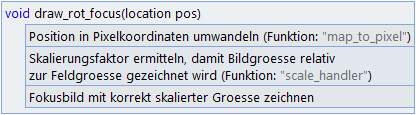
#### draw\_playground

#### scale\_handler



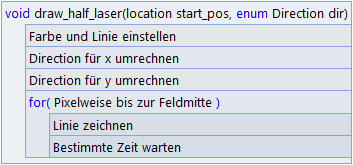
#### draw\_focus

#### draw\_rot\_focus

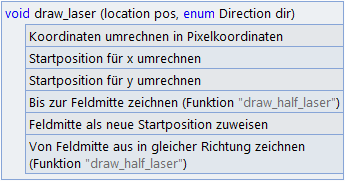


#### draw\_empty\_field

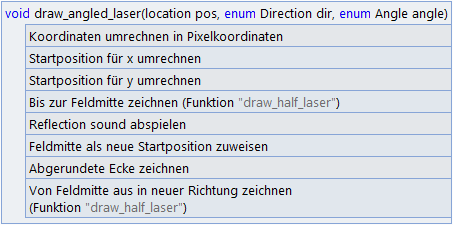
#### draw\_half\_laser



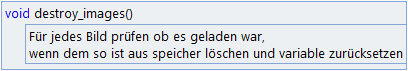
#### draw\_laser



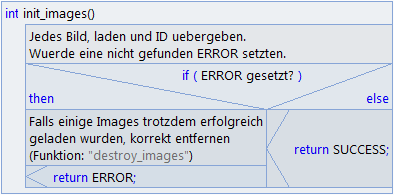
#### draw\_angled\_laser



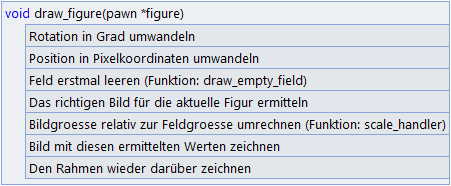
#### destroy\_images



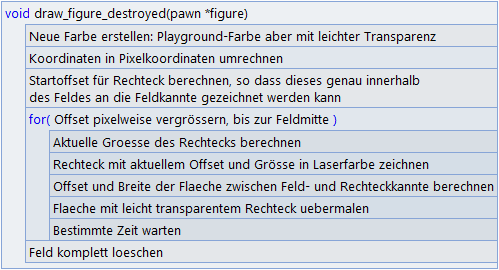
#### init\_images



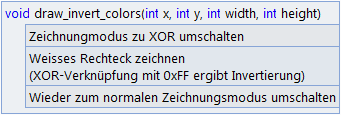
#### draw\_figure



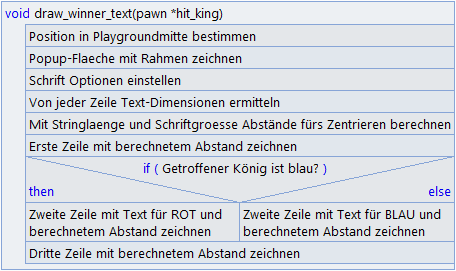
#### draw\_figure\_destroyed



#### draw\_invert\_colors



#### draw\_winner\_text



# Implementation

Die Implementation erfolgte gemäss dem Design. Während der Projektzeit, erweiterte sich das Projekt, und das Design wurde angepasst.

Die Implementation besteht aus den folgenden Dateien:

|  |  |
| --- | --- |
| Dateiname | Funktion |
| LaserChess.c LaserChess.h | Hauptprogramm, Initialisierung, ruft das Spiel auf. In Header-File sind alle Makkros, globalen Variablen, Defines und sonstige Datenstrukturen definiert. |
| Spiel.c Spiel.h | Steuert den Spielablauf |
| Logik.c Logik.h | Führt LASER-Funktion aus, verwaltet sonstige Helper-Funktionen |
| Grafik.c Grafik.h | Verwaltet alle Grafikanwendungen und den Sound (Spielfeld zeichnen, LASER-Strahl zeichnen, Umrechnung zwischen Map und Pixel-Koordinaten, usw.) |
| window.c window.h | Zur Verfügung gestelltes Modul für die Grafikausgabe |

# Zukünftige Erweiterungen

Bei einem Softwareprojekt ist es immer möglich, noch hier und da Verbesserungen einzubauen. Jedoch beschränken wir uns hier vor allem auf Features, die uns beim Spielen des Spiels in den Sinn gekommen sind.

## Spiellogik

Bei der Spiellogik könnte man noch folgendes verbessern:

### Splitter

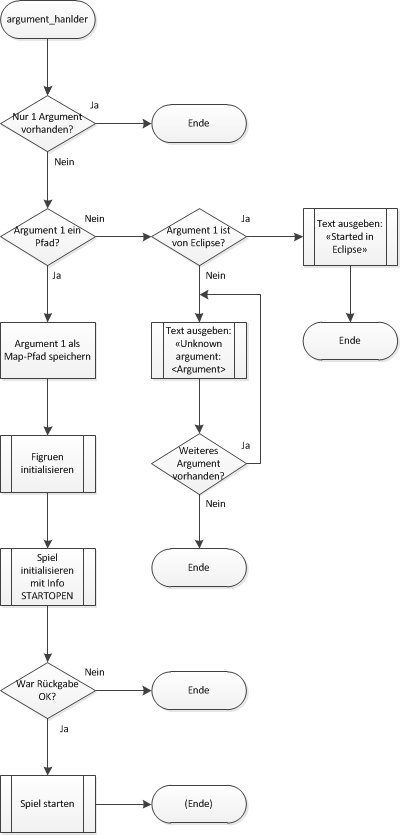
Man könnte die bereits implementierte aber noch nicht eingeschaltete Figur Splitter einschalten. Der Nachteil ist hier, dass wir aufgrund der Struktur der Laserzeichnungsfunktion nicht beide Strahlen, die den Splitter verlassen, gleichzeitig zeichnen können. Zuerst würde der gerade auftretende Strahl (ohne Reflektion) gezeichnet werden und wenn er fertig ist der abgewinkelt austretende Strahl:



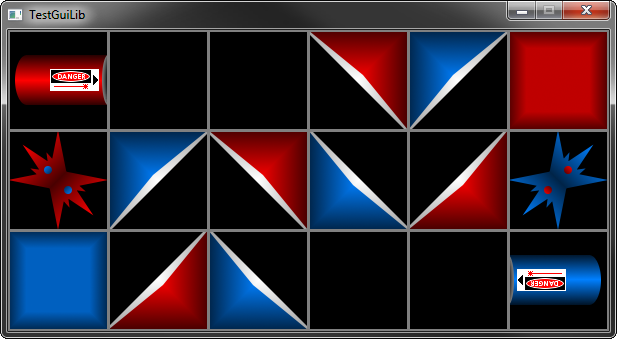
### Dynamische Startparameter

Man könnte die Grösse des Spielfeldes (die Grösse der Felder UND die Anzahl der Felder) dynamisch einstellbar machen, und zwar mit einem Parameter, den man beim Start der Software mitgibt. Und wenn man schon dabei ist, könnte man verschiedene andere Parameter wie z.B. die Spielaufstellung, Farbeinstellungen etc. auch gerade per Startup-Parameter einstellen. Eine derartige Funktion wurde bereits geschrieben und befindet sich im Hauptmodul (LaserChess.c). Die Funktion heisst **argument\_handler()** und bekommt unter anderem dieselben Argumente mit, die eigentlich die **main()** Funktion erhält: **int argc** und **char\* argv[]**. Dann bekommt sie noch einen Zeiger auf die Figuren mit, die sie ihrerseits dann an Funktionen wie **create\_figures()**, **init\_game()** oder **spiel()**.

Das Flowchart dieser Funktion sieht folgendermassen aus:



Um diese Verbesserung in Zukunft einfacher zu implementieren, haben wir uns strikt daran gehalten, keine „Magic Numbers“, also Zahlen grösser als ca. 5, deren Bedeutung nicht völlig logisch ist, einfach so im Code zu schreiben. Sondern wir haben an solchen Stellen immer mit Makros gearbeitet. Hier ist z.B. ein Print-Screen, bei dem im Code NUR die Anzahl Felder in dessen Makrodefinition verkleinert wurde (also **PLAYGROUND\_X\_MAX** und **PLAYGROUND\_Y\_MAX**):



Um auch noch zu zeigen, dass die Feldgrösse und andere Parameter durch Makros verstellbar sind, wird hier noch ein Print-Screen gezeigt:



**Makros in LaserChess.h:**

**PLAYGROUND\_X\_MAX 8** -> ***16*  
PLAYGROUND\_Y\_MAX 6** -> ***12***

**Makros in Grafik.h:**

**FIELD\_SIZE 100** -> ***40*  
LINE\_COL COL\_GREY** -> ***COL\_WHITE*  
PLAYGROUND\_COL COL\_BLACK** -> ***COL\_GREY*  
LASER\_COL COL\_GREEN** -> ***COL\_YELLOW*  
LASER\_WIDTH 2** -> ***4***

### Game over LAN

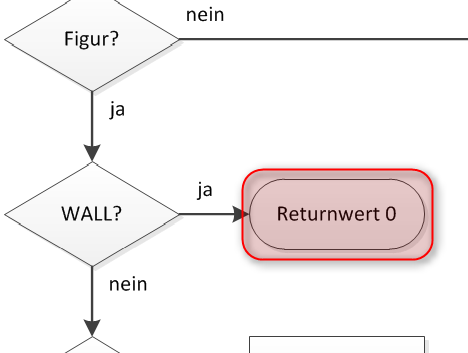
Es wäre möglich, das Spiel über eine Ethernet-Verbindung zu zweit an zwei getrennten Computern zu spielen. Dazu müsste man aber die Spielstruktur etwas anpassen.

### Game-AI

**AI** steht für „Artificial Intelligence“ und bezeichnet im Gamebereich einen computergesteuerten Gegenspieler. Dieser müsste zumindest einmal über ein gewisses Mass an Intelligenz verfügen um zu erkennen, ob sein König im nächsten Zug von einem Laser getroffen wird Danach kann man weitere strategische Entscheidungsmöglichkeiten einbauen.

### Snake-Walls

Um die Schwierigkeitsstufe etwas zu erhöhen, könnte man den Laser durch die Wände hindurchgehen lassen (wie im berühmten Handy-Game *Snake*). Dies wäre programmiertechnisch sogar extrem einfach, da eigentlich nur die Verzweigung WALL der **Laser()** Funktion in eine weitere Laser-Ausführung statt einem Abbruch der Funktion geändert werden müsste (Richtung des Lasers würde ebenfalls bleiben):



## Grafik

### Gif Animationen

Wir sind eigentlich sehr zufrieden damit, wie die grafische Umsetzung jetzt aussieht. Jedoch könnte man noch Gif-animierte Figuren einsetzen, z.B. beim Laser, wenn er schiesst. Oder sogar Gif-animierte Explosionen der getroffenen Figuren einbauen.

### Drag-and-Drop

Anstatt eine Figur zu markieren und dann auf das Zielfeld zu klicken, könnte man mit Drag-and-Drop eine intuitivere Figurverschiebung implementieren. Währenddessen könnte man auch die Figur unter den Mauszeiger „kleben“.

### Lasergeschwindigkeit

Wir hatten bei manchen Computern das Problem, dass der Laserstrahl sich nicht wie gewohnt fortbewegt hat, sondern um ca. Faktor 5 langsamer. Diese Unschönheit, die natürlich auch an den entsprechenden PCs liegen könnte, müsste man noch genauer anschauen.

### Grafikoptionen Real-Time

Extremer Luxus wäre natürlich, wenn die im Abschnitt 6.1.2 beschriebenen Grafikoptionen gleich in einem GUI-Fenster parallel zum Spiel einstellbar wären.

# Anhang

* Pflichtenheft
* Structured Design (CRC)
* Styleguideline