

# Carrerabahn

Weiterentwicklung 2012/2013

Raum 207

2013

# Inhaltsverzeichnis

|   |    |
|---|----|
| 1.Aufgabe                                       | 5  |
| 1.1. Technische Aufgabe                         | 5  |
| 1.2. Visuelle Aufgabe                           | 5  |
| 2.Entwicklungsstand zum Zeitpunkt der Übernahme | 6  |
| 2.1. Grundriss der Bahn                         | 6  |
| 2.2. Reflexsensoren                             | 7  |
| 2.3. UE9  | 7  |
| 2.4. Arduino                                    | 8  |
| 2.5. Ampel                                      | 8  |
| 3.Anfängliche Zielsetzung                       | 9  |
| 3.1. Hardware                                   | 9  |
| 3.2. Software                                   | 9  |
| 3.3. Visualisierung                             | 9  |
| 4.Veränderungen an der Hardware                 | 10 |
| 4.1. Sensorik                                   | 10 |
| 4.2. Platine                                    | 12 |
| 4.3. Schalter                                   | 13 |
| 5.Veränderungen an der Software                 | 14 |
| 5.1. Entwicklungsprozess und Probleme           | 14 |
| 5.1.1. Transition                               | 14 |
| 5.1.2. C, IDEs und veraltete Libraries          | 14 |
| 5.1.3. BackUps                                  | 15 |
| 5.2. Ergebnis                                   | 16 |
| 5.2.1. Programm und UI                          | 16 |
| 5.2.2. Ethernet                                 | 17 |
| 5.2.3. Drucken einer Urkunde                    | 17 |
| 5.2.4. Multithreading                           | 18 |

|   |    |
|---|----|
| 6. Visualisierung   | 19 |
| 6.1. Einführung in PAINT.NET                                    | 19 |
| 6.2. „Colormode“ für das verwendete Bild eines Autos            | 20 |
| 6.3. Tribünenwerbung  | 23 |
| 6.4. Sticker / Neues Raumschild                                 | 24 |
| 6.5. T-Shirts   | 26 |
| 7. Die Entwicklungsschritte, die für uns noch realisierbar sind | 28 |
| 7.1. Hardware   | 28 |
| 7.2. Software   | 29 |
| 7.2.1. Dokumentation + Transition für Zukünftige                | 29 |
| 7.2.2. Multithreading at its best                               | 29 |
| 7.3. Visualisierung   | 30 |
| 8. Weitere Entwicklungsideen                                    | 31 |
| 8.1. Hardware   | 31 |
| 8.2. Software   | 32 |
| 8.2.1. Do it the cloud way                                      | 32 |
| 8.2.2. GUI  | 33 |
| 8.2.3. Turnier Modus  | 34 |
| 9. Arbeitsaufteilung  | 35 |
| 10. Anhang  | 36 |
| 10.1. Quellcode   | 37 |
| 10.2. Drucken einer Urkunde                                     | 58 |
| 10.3. Struktogramm  | 59 |
| 10.4. Bahngrundriss   | 60 |

# 1. Aufgabe

## 1.1. Technische Aufgabe

Im Rahmen des Unterrichts des Faches Technologie war uns Schülern die Aufgabe gegeben worden, möglichst selbstständig an eigenen Projekten zu arbeiten und diese fertig zustellen.

Dabei lag der Schwerpunkt auf die kollaborative Entwicklung in einem Team. Wir mussten intern die Aufgaben verteilen und anschließend diese verantwortungsvoll bearbeiten.

Die Grundlegende Basis wurde im Unterricht der Klassen 11 und 12 gelegt, aber für die einzelnen Projekte musste von noch sehr viel weitere Zeit in eigenständigen Recherchearbeiten investieren, um die Projekte zu einem Abschluss führen zu können.

Das Carrerabahn- Projekt wurde schon von früheren Gruppen entwickelt und die Kernaufgabe von uns lag in der Weiterentwicklung dieser Rennbahn.

## 1.2. Visuelle Aufgabe

Neben der technischen Problemstellung haben wir uns des weiteren der Problematik der Visualisierung in Relation zu dem Projekt gewidmet. Dabei haben wir uns zu Beginn primär auf das Design eines neuen Logos konzentriert, bzw. der Kreation eines Colormods welches auf einem gemalten Auto basiert, hingegeben.

## 2. Entwicklungsstand zum Zeitpunkt der Übernahme

In Raum 207 befindet sich die Carrera-Bahn Pro, diese wurde von dem letzten Jahrgängen zusammengebaut. Sie verfügt über ein autonomes Fahrzeug, welches über Lichtschranken gesteuert wird, auf der Ebene von Python.

Die Spannung ist einstellbar, somit können die Fahrzeuge unterschiedlich schnell fahren und an die jeweilige Altersgruppe der Fahrer angepasst werden. Des Weiteren wurden die Steuereinheiten der Carrera- Bahn neu getaktet um die Geschwindigkeit besser zu kontrollieren; nun ist es sogar möglich mit Standgas zu fahren. Des weiteren wird über Lichtschranken die Rundenzeit der Autos gemessen und gespeichert, diese können nach dem Rennen sogar ausgedruckt werden, was die kleineren Fahrer der Bahn immer sehr beeindruckt.

Das Interface Board wurde via USB an einem Computer mit einem Linux Betriebssystem angeschlossen. Auf diesen Rechner befindet sich das Python Programm, welches die gesamte Rennlogik beinhaltet und das UE9 dementsprechend ansteuert. Nach einem erfolgten Rennen konnte man sich das Ergebnis mit Hilfe eines Druckers ausgeben lassen.



### 2.1. Grundriss der Bahn

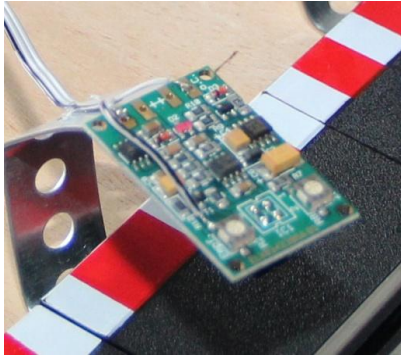
Die Bahn besteht auf 4 Spuren. Sie besteht aus 9 Kurven mit 3 langen Beschleunigungsgeraden. Eine dieser Kurven ist eine 180° Steilkurve.

Weiterhin besitzt die Bahn eine Brücke.

Der Start liegt am Ausgang der letzten Kurve. Im Anhang gibt es einen Grundriss der Bahn (vgl. S. 61) zum derzeitigen Entwicklungsstand.

## 2.2. Reflexsensoren

Um das autonome Fahren realisieren zu können, benötigt man etwas, was die Position des Autos auf der Strecke ermittelt. Dies wurde durch die Reflexlichtschranken realisiert. Diese wurden vor und nach Kurven angebracht, da diesen Stellen der Strecke das Auto beschleunigt bzw. abgebremst werden musste, um ein Optimum an Schnelligkeit zu gewährleisten

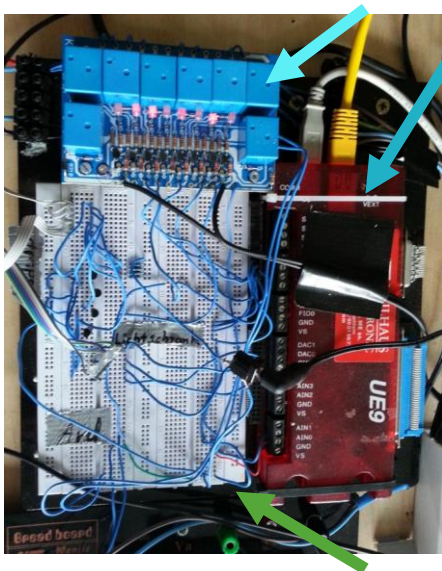


Die Lichtschranken sind dabei über der Bahn angebracht. Passiert ein Fahrzeug die Lichtschranke wird ein Signal ausgegeben.

Sie reagieren auf eine Reichweite von etwa 5 bis 40 Millimeter.

Die Lichtschranken müssen eine Spannung von 8V – 18V erhalten. Die Reaktionszeit kann dabei außer acht gelassen werden, weil sie sich nur um 5 ms handelt und somit nicht wirklich die Messung beeinträchtigt, da es zu keinem Vergleich steht, in dem sich das Auto bewegt.

## 2.3. UE9

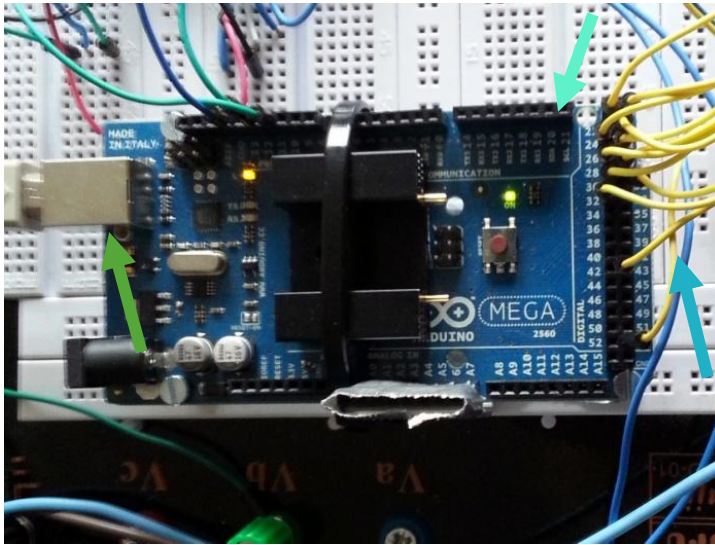


Das UE9 von Meilhaus gehört zur LapJack Gruppe und kann neben USB auch einen Ethernet Anschluss. Es bietet 14 Analoge Eingänge und 23 Digitale Leitungen. Außerdem bietet das UE9 zwei 32bit Zähler.

Das UE9 wird als Interface verwendet; es sind die Sensoren angeschlossen und übergeben den aktuellen Status. Zusätzlich steuert das UE9 auch die Relais zur Spannungsversorgung der einzelnen Bahnen.

Da die Daten verteilt werden müssen, gibt es das Steckboard.

## 2.4. Arduino



Das **Arduino** Mega 2560 ist ein Microcontrollerboard mit 54 digitalen Pins die dem in und Output dienen. Weiterhin besitzt es 16 analoge Inputs und eine USB Schnittstelle.

Am Arduino gehen die **Leitungen von den Sensoren** direkt ein

und werden von dort über **USB** an das UE9 geleitet.

## 2.5. Ampel

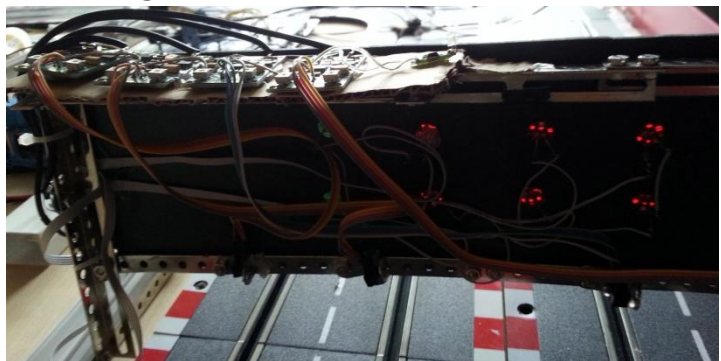


Es existiert eine kleine Startampel, die mit drei roten und einer grünen LED Doppelreihe einen Countdown anzeigt.

Sie ist mit 4 Sensoren ausgestattet- für jede Bahn einen. Über diese Sensoren werden die Rundenzeiten

und die Zieleinfahrten aufgenommen und dem Arduino übergeben.

Die Farben und die Art der Kabel sind Willkürlich gewählt worden.





## 3. Anfängliche Zielsetzung

### 3.1. Hardware

Des weiteren gibt es ein Masse Problem, welches gelöst werden muss, da die Spannung immer wieder absackt, wenn mehrere Fahrzeuge gleichzeitig das Gas betätigen.

Weiterhin sollen die Sensoren überarbeitet und mit einem neuen Kabelbaum versehen werden um die Übersichtlichkeit zu steigern.

### 3.2. Software

Den momentanen Quellcode, welcher auf Python basiert, soll in die Programmiersprache C transferiert werden, damit andere Jahrgänge den Quellcode nachvollziehen können.

Dabei ist der Anspruch, entweder die gleiche Qualität oder eine noch höherwertige Qualität zu erreichen. Das bedeutet die verschiedenen Rennmodi, die Oberfläche, das Drucken von Urkunden, die Kommunikation mit der Rennbahn und eine fehlerfreie Rennlogik in einem ansprechenden Zeitraum komplett nachzubauen bzw. zu verbessern.

Auch sollen vorhandene Fehler, die momentan noch beim autonomen Fahren vorzufinden sind behoben werden.

### 3.3. Visualisierung

Auch die Visualisierung muss überarbeitet werden, da die Momentane Bahnbeschriftung in schwarz-weiß ist, diese soll in Farbe und noch besser gestaltet werden, ebenfalls wollen wir ein großes Plakat anfertigen. Außerdem müssen die Platten, auf der sich die Bahn befindet besser zusammengeschraubt werden, um die Stabilität der Bahn zu sichern. Ebenfalls werden noch Sticker angefertigt und eine Bandenwerbung, auf der die Sponsoren wiederzufinden sind.



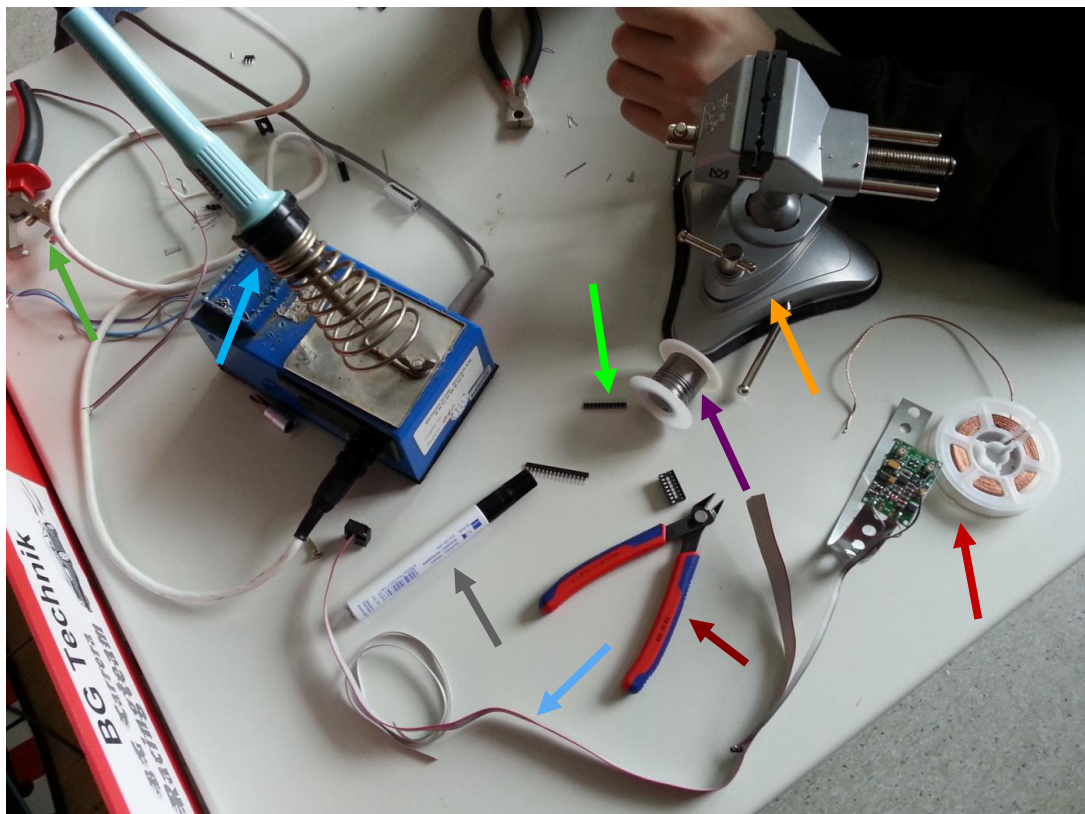
## 4. Veränderungen an der Hardware

### 4.1. Sensorik

Zur Optimierung wurden anfangs die kompletten Sensoren für das autonome Fahren überarbeitet um einen einheitlichen Standard zu erreichen, da vorher die Kabel unterschiedlich an den Sensor angelötet waren und mit Heißkleber isoliert wurden.

Um das autonome Fahrzeug zu Optimieren wurden an sinnvollen Stellen zusätzlich Sensoren angebracht und Ersatz Sensoren angefertigt.

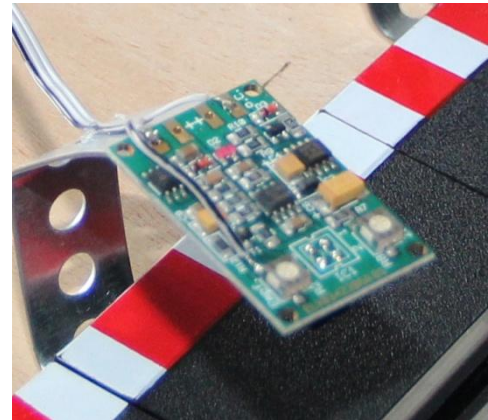
Um Kabel mit Sensoren zu verlöten muss zunächst die richtige Länge der Kabel (in unserem Fall **Flachbandkabel**) gemessen werden. Wenn die Länge bekannt ist, wird das Kabel mit einer **Zange** abgetrennt. Das Flachbandkabel trennt man auf und isoliert es mit einer **Abisolierzange** ab und verdreht die einzelnen Kabel.



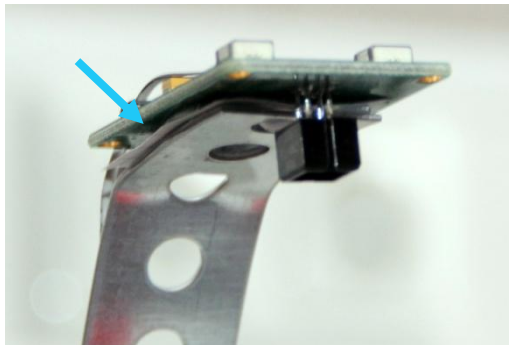
Zur Stabilisation kann man jetzt den Sensor in den **Schraubstock** einspannen. Mit dem **Lötkolben** macht man jetzt **Zinn** heiß und verteilt diesen auf dem zu fixierenden Punkt auf dem Sensor mit dem Kabel.

Wenn dabei etwas schief geht kann man dies mit dem einem **Entlöt** entfernen. Wenn das Kabel Kontakt mit dem Sensor hat wird an das andere Ende des Kabels ein **Stecker** angebracht. Dazu isoliere ich es erst ab, verdrillt und verzinne es.

Dann bricht man sich eine Dreiergruppe Stecker ab und fixiert diese im Schraubstock. Dort stecke ich die Kabel in die Öffnungen und löte sie dort fest. Wenn das alles geschehen ist markiert man eine Seite des Kabels mit einem Filsstift. Das macht es



übersichtlicher. Auf der Abbildung links sieht man einen fertigen Sensor mit Gestell.



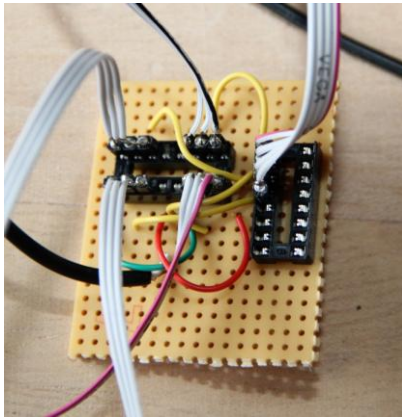
Das Gestell wurde aus Aluminium gebogen und mit Heißkleber fixiert. Davor wurde eine Zwischenschicht **Papier** angebracht um Sensor und Gestell voneinander zu isolieren.

zusätzlichen Sensor angebracht, oft aufgrund seiner zu geringen Geschwindigkeit hängen geblieben ist.

Es wurde an der Brücke einen



## 4.2. Platine



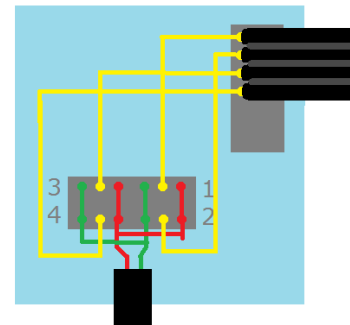
Zusätzlich wurden Sammelpunkte für die Sensoren auf jeder Platine erstellt. Somit ist jede Platten ein einzelnes Modul und kann eigenständig transportiert werden.

Im Gegensatz zu vorher wo jeder Sensor eine eigene Spannungsversorgung und Signalweiterleitung brauchte liefern

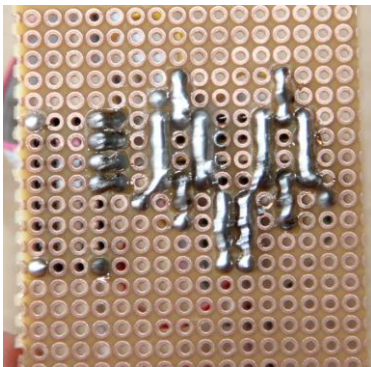
nun die Sammelpunkte die Spannung an alle Sensoren gleichermaßen, zudem werden die Signale gebündelt zu dem Arduino geleitet.

Dies wurde mit einer Platine realisiert.

Rechts ist der Aufbau einer Platine zu sehen. Sie besteht aus zwei Hauptsteckern. In eine diese Hauptstecker werden die Sensorenverbindungen gesteckt. In diesem Fall kann an die Platine

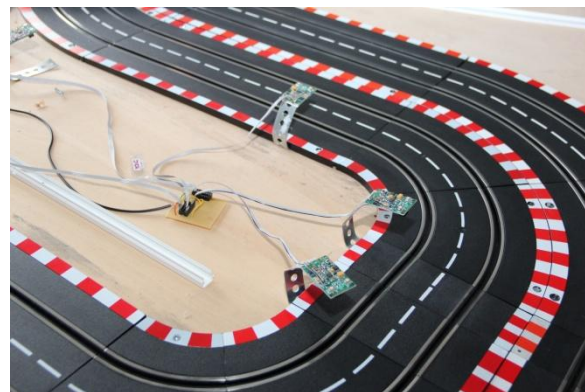


4 Sensoren angeschlossen werden. Gelb sind die entsprechenden Signalkabel, rot die +5V und grün die Masse. Die Daten werden mit einem Flachbandkabel von der Platine geleitet.



Sensoren.

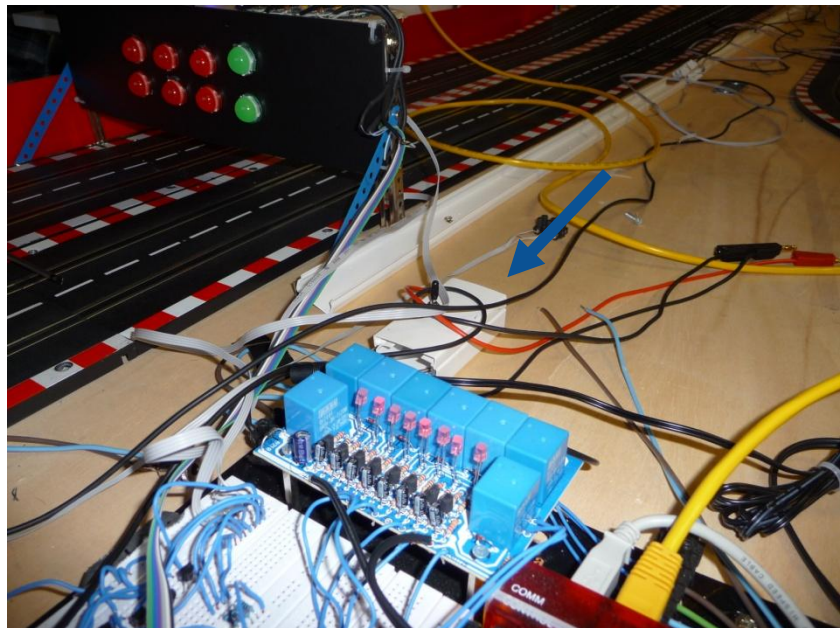
Durch diese Maßnahme konnten die benötigten Kabel wesentlich reduziert werden und die Übersichtlichkeit stark verbessern.





### 4.3. Schalter

Eine weitere  
Neuerung ist  
ein **Schalter**.  
Dieser  
Schalter  
ermöglicht  
den schnellen  
Wechsel  
zwischen  
manuellem  
Fahren und  
fahren mit  
dem  
autonomen  
Fahrzeug.



Er ist direkt an der Startgerade neben den Relais zu finden.

## 5. Veränderungen an der Software

### 5.1. Entwicklungsprozess und Probleme

#### 5.1.1. Transition

Es gab keine umfassende Dokumentation einzelner Abschnitte des Programms in der Dokumentation. Dies führte zu anfangs zu Schwierigkeiten

Fast alle Strukturen und Abläufe mussten im Programmiercode selbst analysiert werden. Dies führte nach einer ersten Inspektion unweigerlich zu einer grundsätzlichen Beschäftigung mit Python.

Anschließend stand dann aber nichts mehr im Weg um mit der Arbeit anfangen zu können.

#### 5.1.2. C, IDEs und veraltete Libraries

Da Python eine objektorientierte Programmiersprache ist während C eine strukturierte Programmiersprache ist gab es einige Probleme. Zwar kann man ein Teil der Objektorientierung durch Elemente wie Strukturen nachbauen, doch die kompletten Vorteile von OOPs kann man nicht ersetzen. Desweiteren ist Python eine sehr minimalistisch gehaltene Programmiersprache, die mit wenigen Zeilen Code viel bewegt. Bei C sind Funktionsblöcke jedoch wesentlich größer und aufwendiger zu entwickeln.

Diese Diskrepanz hat uns dann von dem Weg abgebracht, das Python Programm einfach 1:1 nachzubauen.

Daher kreierten wir völlig eigenständige Anwendungen und übertrugen diese nur noch die eigentlichen Ideen in das neue Programm. Der restliche Teil wurde unabhängig von der ursprünglichen Software entwickelt.

Ein weiteres Problem stellten die Konfiguration und Inkompatibilitäten der IDE dar. Diese wurden mit Hilfe von Microsofts Visual Studio gelöst.

Im Laufe der Zeit gab es einige Probleme, die die Arbeit mehrfach behinderten. So ist die neue Version von VS für die

Entwicklung von modernen Windows Anwendungen entwickelt worden. Für ein Programm, das aber aus Wartungsgründen komplett in C geschrieben werden sollte, erfüllte VS mit den Default Einstellung nicht die Voraussetzung. So konnte der Compiler erst nach einigen sehr langen Konfigurationen dazu gebracht werden, das Programm zufriedenstellend in eine exe-Datei umzuwandeln.

Im Verlauf der Entwicklung stellte sich ein weiteres Problem ein: Viele alte Libraries für C werden nicht mehr aktiv gepflegt und sind damit inkompatibel mit neuen IDEs. So wurden im Entwicklungsprozess die Druckfunktion des Python Programm mit Hilfe von PDF Dateien realisieren. Um dies zu ermöglichen wurde die Haru Library verwenden um schöne und farbenfrohe Urkunden drucken zu können. Doch diese Bibliothek, welche schon veraltet war, basierte wiederum auf zwei weiteren Frameworks auf, die ebenfalls nicht mehr aktiv gepflegt worden. Insgesamt führte es schließlich dazu, dass diese Libraries nicht in die IDE eingebunden werden konnte. Als Workaround entschlossen wir uns, die Druckfunktion anders zu realisieren (vgl. 5.2.3 Ergebnis: Drucken einer Urkunde; Seite 17).

### 5.1.3. BackUps

Im Entwicklungsprozess war es sehr wichtig, dass die Daten immer sicher sind.

Ein Verlust von Programmen, besonders wenn diese Programme auf einer komplett offenen Festplatten liegen, auf der jeder zugreifen kann, ist ein durchaus mögliches Szenario.

Deshalb legten wir von Anfang an einen Dropbox Account für dieses Projekt an, um alle wichtigen Ressourcen automatisch sichern zu können. Damit hat sich auch das Arbeiten auf mehreren Rechnern stark vereinfacht, denn die Dateien werden einfach über die Cloud synchronisiert. Später kam auch noch ein GitHub Account hinzu, um das Arbeiten mit Externen einfacher zu gestalten.

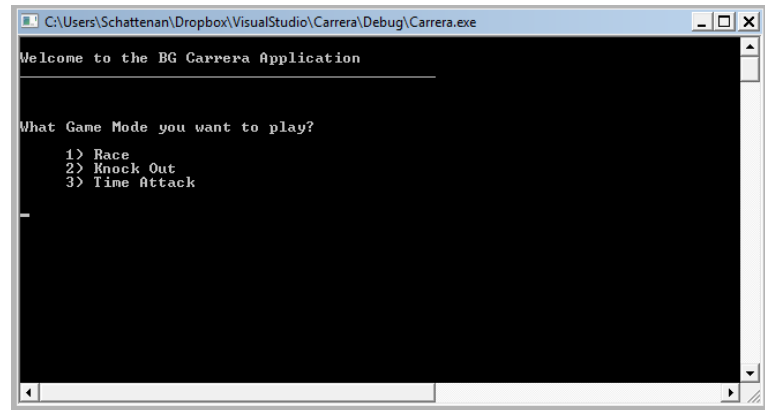
## 5.2. Ergebnis

### 5.2.1. Programm und UI

Das Programm ist zu großen Teilen ein Kommandozeilen-Programm. Zwar sieht es auf den ersten Blick nicht sehr einladend

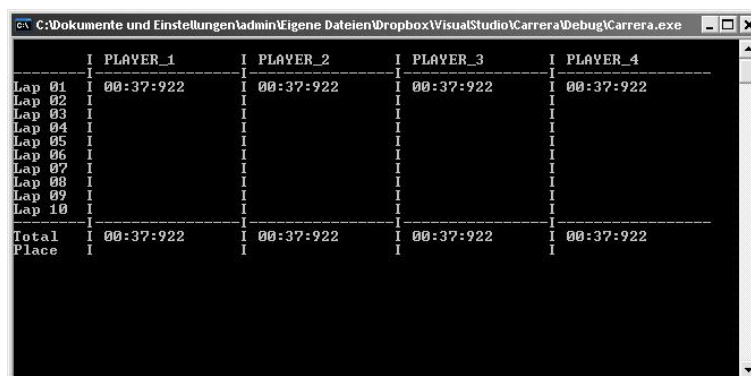
aus, aber dafür ist UI sehr einfach gehalten. Die Abfragen sind klar formuliert und das Programm führt einem

durch den gesamten Prozess zum Starten eines Rennens.



Es muss ein Rennmodus ausgewählt werden und zusätzlich z.B. eine Maximalzeit bzw. eine maximale Rundenanzahl gewählt werden. Anschließend wählt man die Anzahl der Spieler und deren Namen aus. Damit ist der Prozess abgeschlossen und das Rennen kann beginnen.

Im Fenster wird eine Zeittafel mit allen Rundenzeiten der Spieler angezeigt. Die Rundenzeiten werden im Abstand von wenigen Millisekunden immer wieder aktualisiert. Am unteren Ende findet man die Gesamtzeit und am Ende eines Rennens



The screenshot shows a Windows command prompt window titled "C:\Dokumente und Einstellungen\admin\Eigene Dateien\Dropbox\VisualStudio\Carrera\Debug\Carrera.exe". It displays a table of race results for four players. The table has columns for the player name and their lap times. The data is as follows:

|        | PLAYER_1  | PLAYER_2  | PLAYER_3  | PLAYER_4  |
|--------|-----------|-----------|-----------|-----------|
| Lap 01 | 00:37:922 | 00:37:922 | 00:37:922 | 00:37:922 |
| Lap 02 |           |           |           |           |
| Lap 03 |           |           |           |           |
| Lap 04 |           |           |           |           |
| Lap 05 |           |           |           |           |
| Lap 06 |           |           |           |           |
| Lap 07 |           |           |           |           |
| Lap 08 |           |           |           |           |
| Lap 09 |           |           |           |           |
| Lap 10 |           |           |           |           |
| Total  | 00:37:922 | 00:37:922 | 00:37:922 | 00:37:922 |
| Place  |           |           |           |           |

die Platzierung. Die Tabelle ist dynamisch generiert und passt sich der maximalen Rundenzahl an. Aus

praktischen Gründen wurde ein Limit von 99 Runden festgelegt.

Im Anhang (vgl. Seite 60) befindet sich ein Strukturgramm, welches die allgemeine Funktionsweise des Programms zeigt.



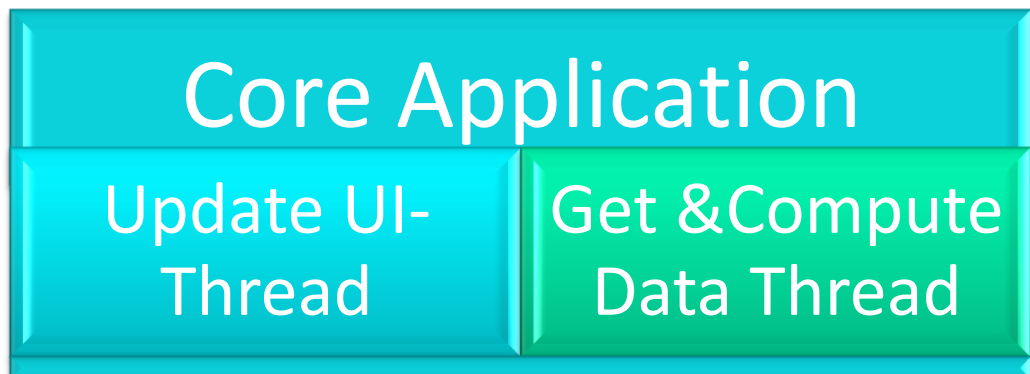
### 5.2.2. Ethernet

Eine grundlegende Verbesserung gegenüber der früheren Anwendung ist die Benutzung von Ethernet als Kommunikationsmedium. Ethernet besitzt den Vorteil einer wesentlich höheren Bandbreite, geringere Latenz, keine zusätzlichen Treiber und ist damit praktisch OS unabhängig als auch besteht die Möglichkeit, von jedem Rechner im Netzwerk als auch von externen Netzen (nicht empfohlen aus Gründen der Latenz) die Rennbahn steuern zu können. Das leitet auch über zu einer neuen flexibleren Ansteuerung der Rennbahn durch Gerätetypen wie Smartphones und Tablet-PCs (vgl. Seite 31).

### 5.2.3. Drucken einer Urkunde

Das Drucken wurde mithilfe eines Workarounds gelöst. Nach Abschluss eines Rennens werden die Renndaten in eine CSV-Datei geschrieben und anschließend mit Hilfe einer PHP Seite die Urkunde in einem Browser angezeigt. Anschließend wird diese dann über das Druckmenü des Browsers ausgedruckt. Der Vorteil dieser Lösung ist, dass sie wie ein Modul fungiert, welches man leicht austauschen bzw. verbessern kann. Durch die vielen Grafik- und Designtechnologien von CSS3 kann man zusätzlich ein visuell beeindruckenderes Ergebnis erzielen als wenn man Urkunden mit PDF Bibliotheken erstellt hätte. Im Anhang(vgl. Seite 59) findet man den allgemeinen Aufbau der CSV-Datei.

#### 5.2.4. Multithreading



Eine weitere sehr wichtige Verbesserung zum ursprünglichen Programm ist die Verwendung von mehreren Threads.

Die Rennsoftware stellt eine sehr zeitkritische Anwendung dar, in der selbst wenige Millisekunden über Sieg oder Niederlage entscheiden können. Bei dem Python Programm gibt es in dem Bezug ein Bug, dass wenn zwei Spieler fast gleichzeitig das Rennen abschließen beide Spieler keine Platzierung bekommen. Um solche (und andere) Probleme zu verhindern arbeitete man in dem neuen Programm mit Threads (um die Genauigkeit zu optimieren) gearbeitet. Bei nur einem Thread werden zuerst alle Sensoren abgefragt, dann die Berechnung der Rennlogik durchgeführt und anschließend die Oberfläche aktualisiert. Es kann durchaus passieren, dass die Rundenzeiten nicht sehr genau sind.

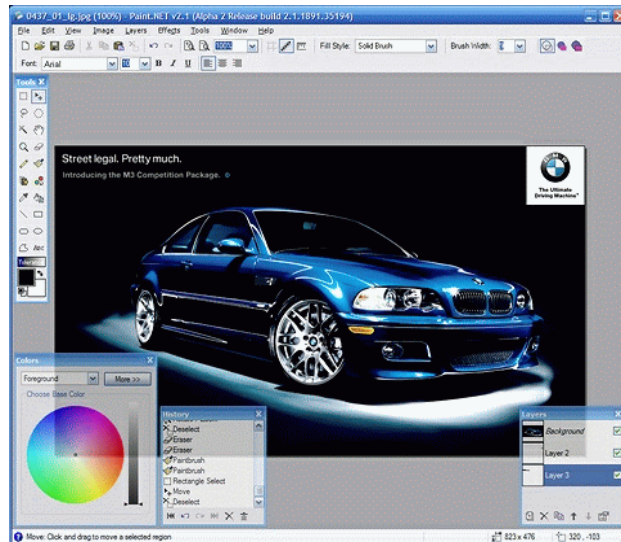
Bei der neuen Lösung übernimmt ein Thread die Aufgabe, permanent die Oberfläche zu aktualisieren. Dieser Prozess ist sehr Zeitintensiv und würde pro Durchgang zu lange dauern, um eine sehr genaue Zeiterfassung im Millisekundenbereich zu ermöglichen.

Ein zweiter Thread übernimmt die Aufgabe der Sensorabfrage und Datenauswertung. Damit bekommt man wesentlich genauere Daten, die bei einem Close Finish einen großen Unterschied ausmachen könnten.

## 6. Visualisierung

### 6.1. Einführung in PAINT.NET

Paint.NET ist eine kostenlose Bildbearbeitungs-Software, welche an der Washington State University mit der Unterstützung von Microsoft entwickelt wird und anfangs lediglich als Ersatz für „MS Paint“ erdacht wurde.



Das besondere an Paint.NET ist, dass sie trotz dem Dasein als Freeware dennoch über Funktionen von teureren, kostenpflichtigen Produkten verfügt, wie bspw. die Ebenenfunktion, mit der man bereits vorhandene Bilder in den Vorder- oder auch

Hintergrund verschieben kann und ebenfalls über ihre Transparenz als auch Überlagerungsart entscheiden kann. Somit sind verschiedene Möglichkeiten gegeben eine vorher ausgewählte Fläche mit dem Schleier einer Farbe zu überziehen, bspw. um ein Auto auf graphischer Ebene zu lackieren.

Dies war lediglich ein Beispiel von vielen Funktionen, die dieses Programm bietet, weitere werden später näher beschrieben.

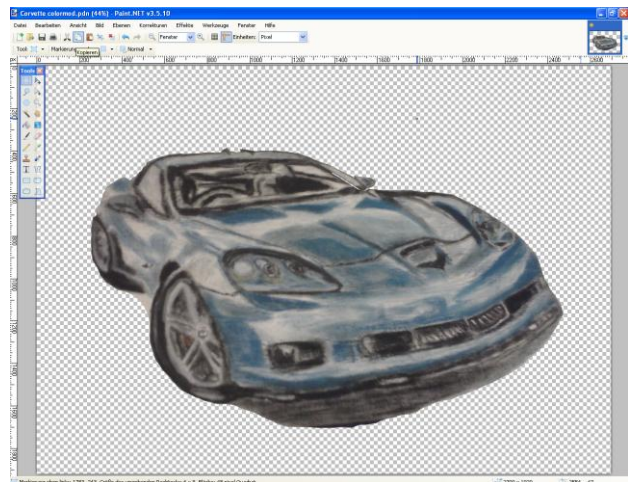
Insgesamt lässt sich sagen, dass Paint.NET bestens für die verschiedenen Aufgaben, die in diesem Projekt aufgetreten sind, zu bearbeiten, meist auf einen verhältnismäßig schnellen Weg.

Bei den folgenden Erklärungen werden die weiteren Tools gelegentlich auf den Screenshots ausgeblendet oder eingeblendet, diese sind jederzeit unter dem Reiter „Fenster“ ein- und ausblendbar.

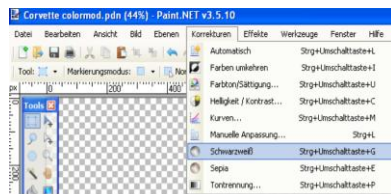
## 6.2. „Colormode“ für das verwendete Bild eines Autos

Auf den folgenden Seiten wird ein Standard Arbeitsprozess veranschaulicht, welcher helfen soll mit dem Programm umzugehen.

Als Basis wird in dem Projekt das Bild eines Autos verwendet, welches zuvor auf Ölfarbenpapier mit Pastellkreide gemalt wurde (deswegen der getupft wirkende Farbauftrag).



Von hier aus geht es zum nächsten Schritt, dem Bild die Farbe entnehmen (damit es neu eingefärbt werden kann, ohne das die gegebene Grundfarbe den ausgewählten Farbton beeinflusst). Dazu geht man lediglich auf den Reiter „Korrekturen“, und wählt in

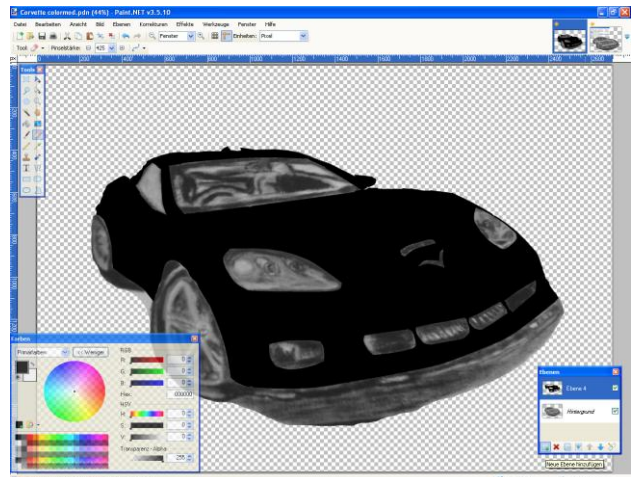


Folge davon den Befehl „Schwarzweiß“ aus (ebenfalls möglich durch die Tastenkombination „Strg“, Umschalttaste und „g“). Nun ist das Auto schwarzweiß gefärbt.

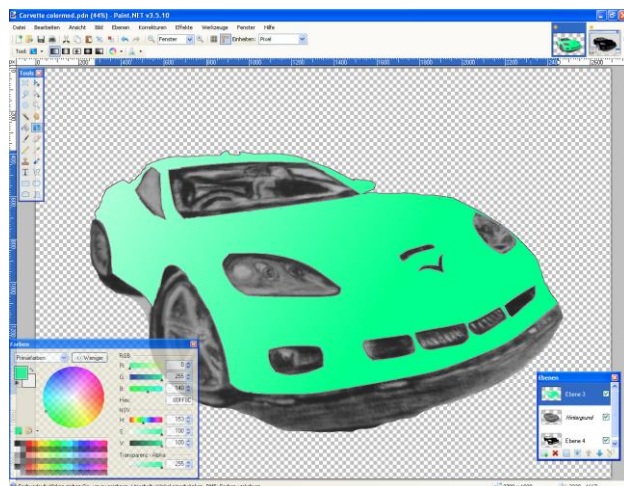
Im Anschluss werden die Ebenen benötigt, zu Beginn lediglich um uns eine Grundlage zugeben, mit der schnell eine Auswahl des zu färbenden Teil vorhanden ist, anstatt diese immer wieder aufs neue zu markieren (dieser Schritt ist nicht bedingt notwendig, er wird lediglich als Zeitersparnis verwendet). Über die „F7“ Taste lässt sich das „Ebenenfenster“ ein- und ausblenden. In diesem klicken wir auf den Befehl „Eine neue Ebene hinzufügen“.

Im Anschluss haben wir eine neue, von der ersten Ebene, welche in Paint.NET als kursiv geschriebenen

„Hintergrund“ bezeichnet wird, geschaffen (jede Ebene ist von den anderen absolut). Als letzten Schritt in dieser Phase wählen wir in dem „Tools“ Fenster (über „F5“ aufrufbar) den Befehl „Freihand“ aus und unterteilen auf der neu geschaffenen Ebene das metallene Gehäuse ohne Reifen, Fenster und Lichter, wonach der nun geschaffene Bereich mittels des Farbeimers (Hotkey: „F“) schwarz gefärbt wird.



Nun ist die Grundlage für unseren „Colormod“ geschaffen. In dem jetzigen Schritt, wird nur noch erklärt, wie dieser zu nutzen ist. Hierbei wird eine weitere Ebene benötigt. Diese wird sogleich geschaffen. Um fortzufahren, muss nun auf unserer sekundären Ebene das schwarze Gehäuse des Autos über das Zauberstab-Tool markiert (Hotkey: „S“). Im Anschluss wechseln wir wieder zu unserer dritten Ebene. Auf dieser ist immernoch die Fläche des Gehäuses markiert. Nun wird das Farben-Fenster via eines Drucks



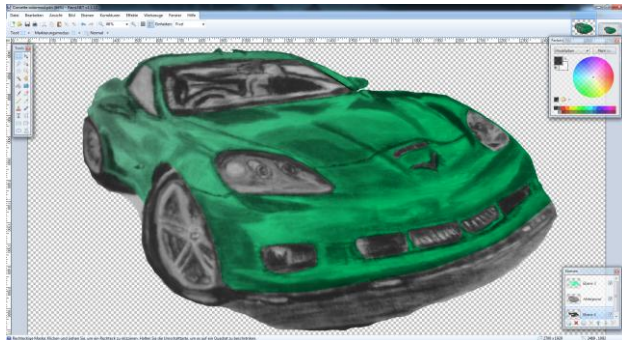
auf „F8“ geöffnet. Jetzt verwenden wir den Farbverlauf Befehl im Tool-Fenster (andernfalls ist der dazugehörige Hotkey: „G“). Bei der Benutzung ist zu beachten, dass die Farbe im Ausgangspunkt den reinsten Wert besitzt, und in die Richtung, in welche

man den Befehl zieht immer heller wird.



Da allerdings das Bild so noch nicht wirklich den anfangs gegebenen Details entspricht, muss im Anschluss die Ebeneneigenschaft der dritten Ebene noch verändert werden.

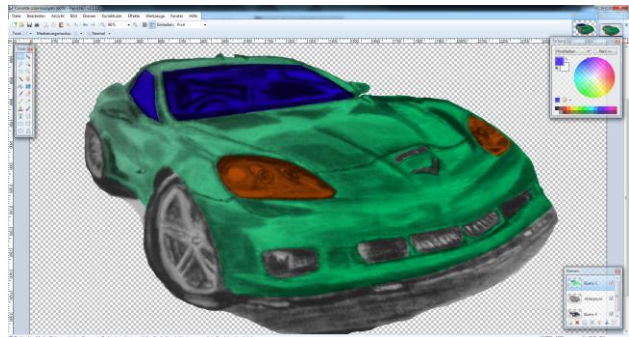
Diese wird über den Hotkey „F4“ geöffnet. Dort verändern wir den Modus von „Normal“ zu „Multiplizieren“ (Sollte Interesse vorhanden sein, kann desweiteren die Transparenz unter „Deckfähigkeit“ in einem Intervall zwischen 0...255 eingestellt werden).



Im Anschluss wurden sämtliche Konturen übernommen (ACHTUNG: die 2. Ebene muss ausgeblendet sein bzw. über die Pfeiltasten im Ebenenfenster eine geringere Wertigkeit als

die der anderen Beiden zugeordnet werden.

Die selben Schritte wie für das Gehäuse des Autos können desweiteren auch für andere Flächen angewendet werden, bzw. der Grundumriss (das schwarze Gehäuse, nach der Anleitung Ebene 2) kann auch über die Reifen gehen, damit bspw. die Felgen auch problemlos ausgewählt und gefärbt werden. Der individuellen Wahl steht dabei ein enormer Freiraum, wodurch jeder die Färbung seines Geschmacks auf die Fläche seiner Wahl anwenden kann. Zum Abschluss noch ein Screenshot des Autos, mit fertiger Färbung.



### 6.3. Tribünenwerbung

Um die Carrera-Bahn an sich noch visuell aufzurüsten, wird noch eine Tribünenwerbung implementiert, welche auf 2 Autos, welche beide durch den zuvor vorgestellten „Colormode“ entstanden sind, basiert. Neben den beiden Rennfahrzeugen kam noch eine gekreuzte Rennflagge hinzu, welche ebenfalls durch Paint.NET mittels eines Schachfeldes an einem schwarzen Stab, welche mehrfach die Auswirkungen verschiedener Effekte unter dem „Verzerren“-Reiter erfahren durfte (hauptsächlich wurden „Drehen...“ und „Gewölbt...“ dafür verwendet). Davon abgesehen befindet sich das Emblem der Beruflichen Schule Eschwege in der Mitte der Rennflaggen über den Autos. Das Emblem dient hierbei lediglich dem Zweck eines Beispiels und soll noch als Dank durch Emblems der Sponsoren, welche unsere Schule bei den Projekten Unterstützung gegeben haben, ersetzt werden (ggf. werden einige der Schulemblems noch aus repräsentativen Zwecken beibehalten). Zum Schluss wurde noch ein Schwarzweißer Hintergrund mittels der Farbverlauf-Befehls eingefügt, um die Tribünenwerbung abzurunden.

Der Prototyp der Tribünenwerbung ist weiterhin frei veränderbar als PDN-Datei für zukünftige Zwecke verfügbar unter dem Namen „Tribünenwerbung.pdn“.





## 6.4. Sticker/ Neues Raumschild

Um noch weiterhin für das Projekt „Racing Team“ zu werben, hat unsere Gruppe sich überlegt Sticker (siehe Abbildung 8) zu kreieren. Sollte sich nun jemand die Frage stellen „Warum habt ihr euch gerade für Sticker entschieden?“, liegt die Antwort klar auf der Hand. Sie sind klein, erfüllen den Zweck einer grafischen Werbung, und kommen, explizit bei jüngeren Publikum, um Längen besser an als jegliche Flyer oder Ähnliches. Davon abgesehen erhöhen die Sticker nicht nur ausschließlich die Bekanntheit des Racing Team Projekts, sondern nebenbei auch noch insgesamt die der Schule, wodurch Kinder/Teenager eventuell auch schon einen Einfluss für die Schulwahl bekommen.

Desweiteren machen Sticker allgemein einen hochwertigeren Eindruck, unter anderem auch bei Firmen, welche die Projekte unterstützen wollen, da diese dabei sehen, dass der Wille eine höhere Bekanntheit zu erreichen und auch die Projekte fortzuführen vorhanden ist.

Die Sticker selbst bestehen aus den zuvor bei der Tribünenwerbung

vorgestellten

gekreuzten

Rennflaggen, welche

durch Flammen

umrandet das

Zentrum und

Hauptaugenmerk

darstellen. Dabei

wirbt Das mittlere

Emblem noch mit auf

den Flammen

geschriebenen

Worten für das

Racing Team und

gibt nebenbei noch die Information, dass es von dem 13.

Jahrgangs des Beruflichen Gymnasiums erarbeitet wird. Neben

dem Schriftzug sind in schwarz noch Tribals auf die Flamme

bearbeitet wurden, welche der Zierde dienen. Dazu kommen noch

Banner die über und unter der Flamme liegen, der obere weist

nochmals expliziter mit den Worten „Racing Team BG 13“ auf das

Projekt an sich hin, der untere Banner hingegen trägt die



Aufschrift „Trust in our work!“ (zu deutsch: „Vertraut in unsere Arbeit!“), was dem Leser den Gedankengang übermitteln soll, dass dieser dem Projekt eine Chance gibt und einen viel versprechenden Eindruck hinterlassen soll.

Der Hintergrund ist ein Farbverlauf von schwarz zu weiß, um den Fokus auf den Frontgrund zu unterstützen (weiß wäre zu unscheinbar, der Übergang soll damit zwar auffallen, allerdings dennoch im Gegenteil zu der Front einen schlichten Eindruck geben).

Die Farben auf dem Sticker wurden mit dem Gedanken gewählt zwar ins Auge zu stechen, jedoch ohne den Beobachter zu erschlagen. Die Verläufe des Lichts dienen dabei als Deutung der Leserichtung, da helle Elemente im Normalfall zuerst wahrgenommen werden.

Von der Verwendung der Grafik als Sticker abgesehen, dient selbiger noch als neues Schild des Raums 207, in welchem die Carrera-Bahn befindet. Der einzige Unterschied besteht darin, dass die Aufschrift des unteren Banners „Raum 207“ angibt.

## 6.5. T-Shirts

Um als Gruppe unser Projekt professioneller vertreten zu können, haben wir uns dazu entschieden T-Shirts zu machen, die uns zum einen eine optische Zusammengehörigkeit verschafft, und zum anderen auch die Personen bei Fragen direkt auf die Ansprechpartner hinzuweisen. Bei den T-Shirts bestehen zur Zeit zwei Prototypen, kommen wir zu dem ersten.

Bei dem T-Shirt handelt es sich um ein gewöhnliches T-Shirt in dem Farbton ultramarine.

Auf der linken Brust befindet sich das Emblem der Beruflichen Schulen Eschwege, darunter steht in weißer Schrift (weiß wurde gewählt, da es sich gut von dem dunkelblauen Farbton abhebt) der Name des Trägers. Auf dem Rücken hingegen befindet sich das Emblem unseres Racing Teams, welches ebenfalls die Basis des Stickers symbolisiert.

Neben der ersten Version ist die sekundäre allgemeiner gehalten, so dass neben dem Racing Team auch die anderen Projektgruppen es benutzen können.



Bei dem T-Shirt hingegen handelt es sich um ein monoton schwarzes T-Shirt.

Das Emblem bei dieser Version ist ein vollkommen neues, es zeigt ein Zahnrad (das Zahnrad wurde gewählt, da es sich um ein bekanntes, technisches Element handelt), welches an jedem der



Innenräume ein hellblaues Licht ausstrahlt (dieses wiederum soll symbolisch bedeuten, dass jedes Zahnrad ineinander passt, was die Unterstützung in den Gruppen, bzw. auch gruppenübergreifend darstellt). Desweiteren sind die Zahnräder mit goldenen Flügeln versehen (diese stehen für Erfolg und sollen auch einen optimistischen Zukunftsblick darstellen, dass man in einen guten Job „aufsteigt“, ergo beflügelt ist).

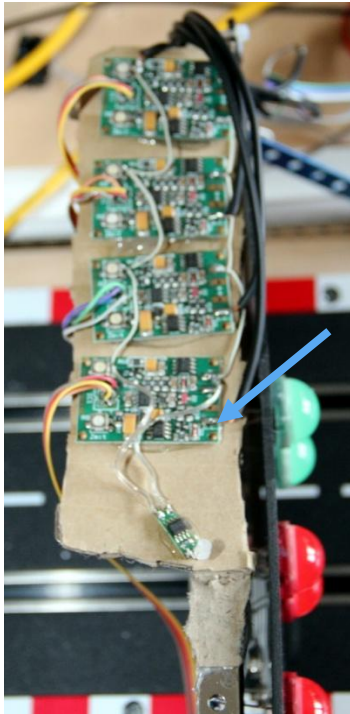


Auf der Brust befindet sich Das Emblem mit den Banneraufschriften: „Call me:“ (zu deutsch: „Nenne mich:“) und den jeweiligen Namen des Trägers, hierbei wurden die hellblauen Lichter ausgeblendet. Auf dem Rücken hingegen besitzen die Banneraufschriften die Worte „Trust me,“ (zu deutsch: „Vertrau mir,“)

und „I am an engineer!“ (zu deutsch: „Ich bin ein Ingenieur!“). Der Hintergrundgedanke hierbei war den Betrachtern der Shirts zu vermitteln, dass es sich bei dem Träger um eine fähige Person handelt (ggf. auch, dass dieser den Job eines Ingenieurs nachiefert und diesen später als Beruf ausführen mag).

## 7. Die Entwicklungsschritte, die für uns noch realisierbar sind

### 7.1. Hardware



Zum derzeitigen Standpunkt, müssen wir die Sensoren noch einstellen und den Fehler der Stromversorgung finden.

Außerdem müsste die Start- und Zielvorrichtung überarbeitet werden. Dabei müssten Schönheitsfehler, wie die Pappe auf der die Sensoren angebracht sind ersetzt werden; dies könnte z.B. ein kleines Holzstück oder einer Aluminiumvorrichtung geschehen.

An der Ampel könnte man auch die Kabelform und die Kabelfarbe überarbeiten.

Das jeweilige Signalkabel der Sensoren müsste passend zum Arduino verbunden und geprüft werden.

Die alten Kabel wollen wir noch „säubern“ und zusammenrollen, damit sie wiederverwendet werden. Dieses „aufarbeiten“ zählt für alle wiederverwendbaren Gegenstände.

Von Vorteil wäre es, wenn wir für den nachfolgenden Jahrgang noch die Kabel beschriften würden.

Außerdem sollte die Dokumentation auf dem neusten Stand gehalten werden um den nachfolgenden Jahrgängen das einarbeiten zu erleichtern

## 7.2. Software

### 7.2.1. Allgemein

In der restlichen Zeit soll erstmal den Code optimiert werden. Im Laufe der Entwicklung haben sich einige Dinge verändert und damit ist auch viel Unrat entstanden. Unnötige Befehle und Funktionen sollen gestrichen werden um die Geschwindigkeit noch weiter zu verbessern. Desweiteren soll auch, nach dem Winterferien die erste Beta Version abgeschlossen werden und das Programm soll auf "Herz und Nieren" von Testen geprüft werden um noch mögliches Verbesserungspotenzial zu finden.

### 7.2.2. Dokumentation + Transition für Zukünftige

Um einen einfacheren Übergang für die nächste Generation an Entwicklern zu ermöglichen, wird eine wesentlich "Code nähere" Dokumentation entstehen, in der im Detail auf einzelne Funktionen und Funktionsabschnitte eingegangen wird um ein möglichst hohen Grad an Hilfestellung und Transparenz zu erreichen. Dadurch soll das nächste Team die Möglichkeit bekommen schon in einer sehr kurzer Zeit mit der Weiterentwicklung beginnen zu können.

### 7.2.3. Multithreading at its best



Angestrebt werden weitere Erweiterung des Multithreadings an mit der Verteilung der Aufgaben auf drei Threads. Ein Thread für die UI, ein zweiter Thread nur für die Abfrage der Sensoren und ein dritter für die Datenauswertung. Damit würde der Loop, der die Sensordaten abfragt, noch kürzer werden und somit würde man noch genauere Daten erreichen.

### 7.3. Visualisierung

Für die Zukunft wird noch eine Umsetzung des T-Shirts, bzw. der Sticker, Tribünenwerbung und Ähnliches angestrebt, und ggf. die Bahn noch durch optische Veränderungen (eventuell auch auf elektronischer Art) aufgerüstet. Diese Veränderungen könnten bspw. eine durch LED angetriebene Bremskurve und Ähnliches sein, jedoch ist dieser Part noch unklar ob er noch umgesetzt werden soll oder nicht. Desweiteren soll noch ein Banner entworfen werden, das der jetzig gestaltete noch nicht den Vorstellungen unsererseits entspricht.



## 8. Weitere Entwicklungsideen

### 8.1. Hardware

Zusätzlich zu den Nacharbeiten wie die Überarbeitung der Start und Zielanlage, könnte man sogenannte Checkpoints einfügen. Checkpoints sind bestimmte Punkte, die im Rennsport passiert werden müssen. An ihnen wird die Zeit zusätzlich aufgenommen, so dass es bestimmte Werte für die Sektoren gibt. In der Industrie wird aus diesen Daten herausgelesen, wie gut das Fahrzeug mit den Kurven- und Beschleunigungsverhältnissen zurecht kommt. Demnach können z.B. die Fahrwerkseinstellungen geändert werden.



Für uns hat das jedoch keine große Bedeutung, trotzdem wäre es interessant ein oder mehrere Autos selber zu bauen. Vielleicht könnte sich dadurch auch das Problem mit den [Kontakten](#) lösen lassen, wenn man diese in dem neuen Fahrzeug

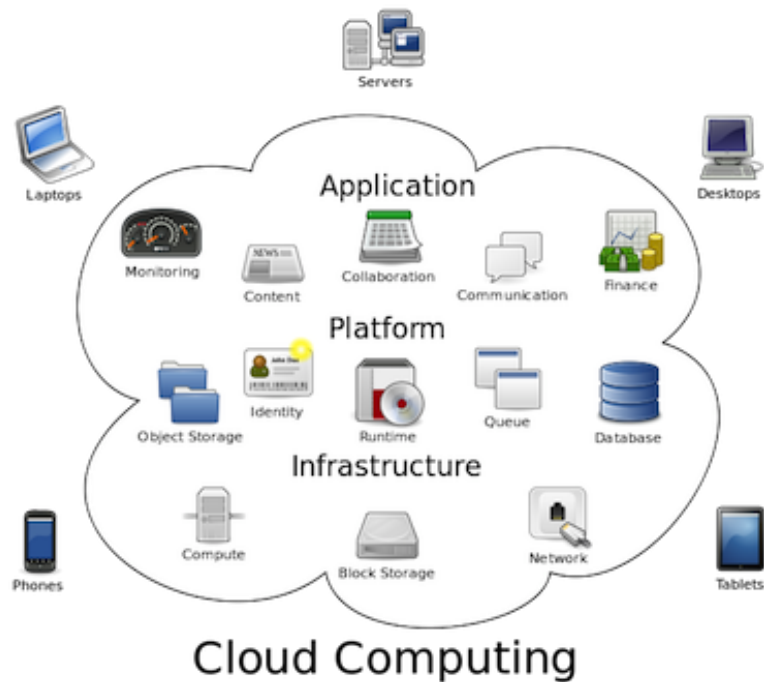
entsprechend verstärkt. Vielleicht könnte man das Bremsen auch in den Autos umsetzen.

Für die Anschaulichkeit könnte man die Kabel unter der Bahn verlegen. Dazu müsste man die einzelnen Stecksysteme entnehmen und die [Querbrücken](#) mit Kerben versehen, die so groß sind, dass das Kabel hinein passt, ohne dass die Bahn angehoben wird.

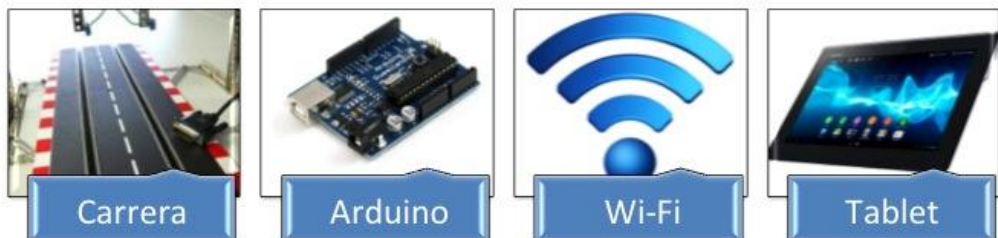


## 8.2. Software

### 8.2.1. Do it the cloud way



In den letzten Jahren hat sich ein alter Trend neu entwickelt: Statt die große Leistung in das Haus eines jeden Menschen zu bringen, werden viele Dinge mittlerweile auf großen Rechenzentren ausgelagert. Ob es sich jetzt um Storage oder um Rechenkraft handelt, am Ende läuft alles auf Seiten der Dienstleister. Der Kunde greift nur noch über kleine und leichte Geräte, welche als Terminal dienen, auf diese Leistungen zu. Damit erreicht man komplette Unabhängigkeit des eingesetzten Betriebssystems als auch der Geräte Typen. Wäre es an sich nicht interessant, dieses System auch für die Carrerabahn zu nutzen?



Die komplette Bahn wäre nicht mehr mit dem UE9 angeschlossen, sondern mit einem Micro Controller (wie z.B. ein Arduino)). Die großen Arduino's wie der Arduino Due sind mit 54 Ports, eine 83 MHz CPU und 96 KB RAM ausreichend

ausgestattet, um die Aufgabe des UE9 erfüllen zu können. Doch man kann sie noch verbessern. Statt einzelne Sensoren von einem Rechner aus abzufragen, kann man den Arduino selbst die Ports abfragen lassen und nur bei einer Veränderung einfach eine Nachricht an dem Computer schicken. Aktuell wird das Ethernetkabel permanent mit TCP Anfragen über den Status von Sensoren überschüttet. Damit ist eine kabellose Kommunikation ausgeschlossen. Doch wenn man eine intelligente Rennbahn besäße, könnte man diesen Traffic auf ein Minimum reduzieren und somit WLAN Verbindungen ermöglichen.

Dies würde bedeuten, dass man nicht mehr einen klobigen alten Rechner mehr benutzen müsste, sondern man könnte ein kleines und leichtes Tablet als Interface verwenden. Besonders bei Präsentation vor Publikum würde so eine Lösung meiner Meinung nach für ein überraschenden WoW-Effekt führen.

#### 8.2.2. GUI

Eine grafische Oberfläche nahm in der Entwicklung keine große Rolle ein, da der Nutzen im Vergleich zu einer Kommandozeilenoberfläche in keinem Verhältnis zum Aufwand stand. Besonders mit dem Anspruch, möglichst alles unter der Programmiersprache C zu realisieren, war es äußerst schwierig, eine GUI zu entwickeln. Das größte Problem stellt der fehlende objektorientierte Ansatz dar, der in der GUI Entwicklung de facto Standard ist. Eine Recherche war sehr aufwändig und stand in keinem Verhältnis zum endgültigen Nutzen. Für zukünftige Entwicklung wäre der Schritt nicht unmöglich. Besondere wenn man bedenkt, dass schon jetzt die Oberfläche in einem eigenen Thread läuft. Ein Austausch dieser Komponente ist also relativ einfach.

### 8.2.3. Turnier Modus

Schon das erste Entwicklerteam hatte die Idee, eine Datenbank für alle Rennen anzulegen. In dieser sollte dann alle Renndaten abgespeichert werden um dann, als Beispiel, eine Bestenliste auszugeben.

Das Problem einer universellen Datenbank ist, dass die Ergebnisse nicht vergleichbar sind. Die Rundenzeiten hängen von vielen Faktoren ab: Zustand der Bahn und der Kontakte, Zustand der Autos, Spannung des Netzteils, usw... . Man müsste für jede mögliche Zusammenstellung dieser Faktoren eine eigene Datenbank erstellen, um Fairness zu wahren.

Doch eine Idee, welche eine Art Vergleich ermöglichen würde, wäre ein Turnier Modus. In diesem könnten z.B. 12 oder 16 Spieler teilnehmen. Es wird in einer Art Ligasystem wiederholt Rennen gefahren bis jeder Spieler einmal in einen Rennen mit jedem anderen Spieler war. Anschließend wird eine Tabelle erstellt, in der die Spieler nach gewonnen Punkten in Spielen und Gesamtzeiten angereiht werden.

Der Vorteil gegenüber einer Datenbank ist, dass sich ein Tuner innerhalb von 90 Minuten absolvieren lässt. Die Chance, dass sich in diesem Zeitraum die Verhältnisse der Strecke und der Wagen rapide verändert, ist recht gering. Wenn also die Spannung des Netzteils eine konstante Leistung liefern würde, hätte man am Ende Daten, die man sehr gut vergleichen könnte.

Man könnte sogar mit der Exportfunktion des Programms eine kleine Webseite mit PHP entwickeln, in der sich jeder die Statistiken nach einzelnen Rennen und Spielern ansehen könnte. Es bräuchte nur einen Parser, der die Daten auswertet und diese ansprechend darstellt.

## 9. Arbeitsteilung

| Gruppenmitglied   | Aufgaben                                   |
|-------------------|--|
| Maximilian Kößler | Hardwareoptimierung                        |
| Tim Schäfer       | Protokoll, Hardwareoptimierung             |
| Mona Schellhase   | Öffentlichkeitsarbeit, Hardwareoptimierung |
| Nils Winkelbach   | Softwareunterstützung, Visualisierung      |
| Björn Wittmann    | Software                                   |

# Anhang

---

## 10.1. Quellcode

```
===== definitions.h =====

//Lists of editable values

#define BUILD "00021 - 2013/01/17"
#define SEPERATOR 44 // ASCII for ',' important for csv export
#define EXPORTPATH "stats.csv" // path to csv-file
#define IMPORTPATH "stats.csv" //@TO-DO: Not yet implemented // path to the webpage certificate
#define DEFAULTIP "192.168.1.209" //default IP for UE9 for TCP connection
#define DEFAUaLTPORT 52360 //default port for UE9 for TCP connection

===== main.c =====

#include "export.h"

int main() {

    char f;
    RACE objectRace;

    initRACE(&objectRace);
    printf("\nBuild : %s \n",BUILD);

    initUI(&objectRace);
    run(&objectRace);
    exportCSV(&objectRace);

    closeConnectionUE9(&objectRace.device.ue9);

    f = getch();
    return 0;

}

===== export.c =====

#include "export.h"

/*

===== stats.csv =====

Mode
Number of Players
Number of Rounds
Playernames
Roundtimes
Ranking
Best round
Total time
```



End signal

```
*/

void exportCSV(RACE *ret)
{
    FILE *datei;
    int i,j;

    if( (datei = fopen(EXPORTPATH,"w")) == NULL )
    {
        printf("Failure accessing stats.csv");
        exit(0);
    }

    // == Game Mode ==
    if(ret->match_Active)
        fprintf(datei,"Rennen");
    if(ret->knockOut_Active)
        fprintf(datei,"Der Letzte fliegt");
    if(ret->timeAttack_Active)
        fprintf(datei,"Time Attack");
    fprintf(datei,"\n");

    // == Number of Players ==
    fprintf(datei,"%d",ret->numberOfPlayers);
    fprintf(datei,"\n");

    // == Number of the max. Rounds ==
    fprintf(datei,"%d",ret->maxRounds-1);
    fprintf(datei,"\n");
    printf("%d \n",ret->maxRounds-1);

    // == Player names ==
    for(i=0;i<ret->numberOfPlayers;i++)
    {
        fprintf(datei,"%s",ret->players[i].playername);
        printf("%d",ret->numberOfPlayers);
        if(i<ret->numberOfPlayers-1)
            fprintf(datei,"%c",SEPERATOR);
    }
    fprintf(datei,"\n");

    // == Round time for each player for each round ==
    for(i=0;i<ret->maxRounds;i++)
    {
        for(j=1;j<ret->numberOfPlayers;j++)
        {
            printf("%d",ret->numberOfPlayers);
            exportTime(datei,ret,j,i);
            if(j<ret->numberOfPlayers-1)
                fprintf(datei,"%c",SEPERATOR);
        }
        fprintf(datei,"\n");
    }

    // == Ranking ==
    for(i=0;i<ret->numberOfPlayers;i++)
    {
        fprintf(datei,"%d",ret->players[i].rank);
        if(i<ret->numberOfPlayers-1)
```

```

        fprintf(datei,"%c",SEPERATOR);
    }
    fprintf(datei,"\n");

    // == Best Round (TBD) ==
    for(i=0;i<ret->numberOfPlayers;i++)
    {
        fprintf(datei,"%d",ret->players[i].bestRound);
        if(i<ret->numberOfPlayers-1)
            fprintf(datei,"%c",SEPERATOR);
    }
    fprintf(datei,"\n");

    // == Total time for each player ==
    for(i=0;i<ret->numberOfPlayers;i++)
    {
        exportTotalTime(datei,ret,i);
        if(i<ret->numberOfPlayers-1)
            fprintf(datei,"%c",SEPERATOR);
    }
    fprintf(datei,"\n");

    //End sequence
    fprintf(datei,"End");

    fclose(datei);

    system(IMPORTPATH); //Opens the website for printing
}

void exportTime(FILE *datei, RACE *ret, int player, int round)
{
    int minuten,sekunden,millesekunden;

    //Getting the time, very similar to function printTime in modes.c
    if(round==0)
        millesekunden = ret->players[player].roundTime[0]-ret->players[player].startTime;
    else
        millesekunden = ret->players[player].roundTime[round]-ret->players[player].roundTime[round-1];

    //Some crazy math nobody understands so don't even try it, you will fail anyway ;- )
    sekunden = millesekunden / 1000;
    minuten = sekunden / 60;
    millesekunden = millesekunden - 1000*sekunden;
    sekunden = sekunden - 60*minuten;

    fprintf(datei,"%02d:%02d.%03d",minuten,sekunden,millesekunden);
}

void exportTotalTime(FILE *datei, RACE *ret, int player)
{
    int minuten,sekunden,millesekunden;

    //Getting the total time, very similar to function exportTime
    millesekunden = ret->players[player].endTime-ret->players[player].startTime;

    //Some crazy math nobody understands so don't even try it, you will fail anyway ;- )
    sekunden = millesekunden / 1000;
    minuten = sekunden / 60;
    millesekunden = millesekunden - 1000*sekunden;
    sekunden = sekunden - 60*minuten;

```

```

        fprintf(datei,"%02d:%02d.%03d",minuten,sekunden,millesekunden);
    }

```

```

===== export.h =====

```

```

#include "modes.h"

```

```

void exportCSV(RACE *ret);

```

```

/*

```

```

The function exports all race data into a csv-File.

```

```

Afterwards it is opening the webpage for viewing and printing the results

```

```

*/

```

```

void exportTime(FILE *datei, RACE *ret, int player, int round);

```

```

/*

```

```

The function prints the roundtime of a specific player's round into a file

```

```

*/

```

```

void exportTotalTime(FILE *datei, RACE *ret, int player);

```

```

/*

```

```

The function prints the total time of a specific player into a file

```

```

*/

```

```

===== modes.c =====

```

```

#include "modes.h"

```

```

void initRACE (RACE *ret) {

```

```

    ret->finished=false;

```

```

    ret->started=false;

```

```

        ret->match_Active=false;

```

```

        ret->knockOut_Active=false;

```

```

        ret->timeAttack_Active=false;

```

```

        ret->activeSensor[0]=false;

```

```

        ret->activeSensor[1]=false;

```

```

        ret->activeSensor[2]=false;

```

```

        ret->activeSensor[3]=false;

```

```

        ret->knockOutPlayerX[0]=0;

```

```

        ret->knockOutPlayerX[1]=0;

```

```

        ret->knockOutPlayerX[2]=0;

```

```

        ret->knockOutPlayerX[3]=0;

```

```

        ret->numberOfPlayers=0;

```

```

        ret->maxTime=999999999;

```

```

    initDEVICE(& ret->device);

```

```

        ret->playerLine=12;

```

```

    initPLAYER(& ret->players[0]);

```

```

        initPLAYER(& ret->players[1]);

```

```

        initPLAYER(& ret->players[2]);

```

```

        initPLAYER(& ret->players[3]);

```

```

        ret->knockOutPlayer=0;

```

```

        strcpy(ret->players[0].playername,"PLAYER_1");
        strcpy(ret->players[1].playername,"PLAYER_2");
        strcpy(ret->players[2].playername,"PLAYER_3");
        strcpy(ret->players[3].playername,"PLAYER_4");
    }

    void match (RACE *ret) {

        nextRound(&ret->players[ret->playerLine],ret->currentTime); //sets new round

        if(ret->players[ret->playerLine].rounds == ret->maxRounds) //player has finished the race
        {
            setPower( &ret->device,(ret->playerLine),false ); //Power off for player

            ret->knockOutPlayer++; //one less player active
            ret->players[ret->playerLine].finished=true;
            ret->players[ret->playerLine].endTime=clock();
            ret->players[ret->playerLine].rank = ret->knockOutPlayer; //ranking, count up 1-4

            // ----- Checks all players ----- \\

            if(ret->knockOutPlayer==ret->numberOfPlayers) //If all player have finished, the
race has ended
            {
                ret->finished=true; //race has ended
            }
        }

    }

    void knockOut (RACE *ret) {

        int i;

        if(!ret->players[ret->playerLine].finished) //If player is finished, he won't be able to drive
more rounds
            nextRound(&ret->players[ret->playerLine],ret->currentTime);

        if(ret->players[ret->playerLine].rounds>0) { //All player need at least one round

            ret->knockOutPlayerX[ret->players[ret->playerLine].rounds-1]++; //counting, how
many players have reached a specific lap number

            //If a player is the last; for example: 4th in the first round, 3rd in the second round,
etc..
            if(ret->knockOutPlayerX[ret->players[ret->playerLine].rounds-1]==ret-
>numberOfPlayers-ret->players[ret->playerLine].rounds+1)
            {
                ret->knockOutPlayerX[ret->players[ret->playerLine].rounds-1]=0;
//counter is going to be resseted
                ret->knockOutPlayer++; //one player out

                ret->players[ret->playerLine].finished=true;
                ret->players[ret->playerLine].endTime=clock();
                for(i=ret->players[ret->playerLine].rounds;i<ret->maxRounds+1;i++) //all
future rounds are going to be set (important for export)
                    ret->players[ret->playerLine].roundTime[i]= ret->players[ret-
>playerLine].endTime;
            }
        }
    }

```

```

        setPower( &ret->device,ret->playerLine,false ); //Turn of the player's track
        ret->players[ret->playerLine].rank=ret->numberOfPlayers-ret->players[ret-
>playerLine].rounds+1; //Set rank, counting down from 4 to 1

        if(ret->knockOutPlayer==ret->numberOfPlayers) //If all players have been
knocked out, the game is finished
            ret->finished=true;
    }

}

}

void placingTimeAttack(RACE *ret) {

    int i;
    int runde[4]= {ret->players[0].rounds,ret->players[1].rounds,ret->players[2].rounds,ret-
>players[3].rounds};
    int win[4]={0,0,0,0};
    int platz[4];
    int j,k=0,m;

    //This is a genric sort feature, Part 1
    for(i=0;i<ret->numberOfPlayers;i++)
    {
        for(j=ret->players[i].rounds;j<ret->maxRounds+1;j++) // sets all missing lap
times to endTime (important for export)
            ret->players[ret->playerLine].roundTime[j]=clock();
        //n*n
        for(j=0;j<ret->numberOfPlayers;j++)
        {
            if(runde[i]>=runde[j])
                win[i]++; // counts how many times a player has more rounds
finished than another
        }
    }

    //This is a genric sort feature, Part 2
    for(i=0;i<ret->numberOfPlayers;i++) //Place
    {
        m=k;
        for(j=0;j<ret->numberOfPlayers;j++) //Player
        {
            if(win[j]==ret->numberOfPlayers-i)
            {
                ret->players[j].rank=m+1;
                platz[j]=m+1;
                k++;
            }
        }
    }
}

void run (RACE *ret) {

    HANDLE hThread[2];

    countdown(ret); //countdown

    if(!ret->started)
    {
        printf("Countdown failed, get used to it");
    }else{

```

```

        //Set all tracks active
        if(ret->numberOfPlayers>0){setPower( &ret->device,0,true ); ret-
>players[0].roundTime[0]=clock(); }
        if(ret->numberOfPlayers>1){setPower( &ret->device,1,true ); ret-
>players[1].roundTime[0]=clock(); }
        if(ret->numberOfPlayers>2){setPower( &ret->device,2,true ); ret-
>players[2].roundTime[0]=clock(); }
        if(ret->numberOfPlayers>3){setPower( &ret->device,3,true ); ret-
>players[3].roundTime[0]=clock(); }

        hThread[0] = CreateThread(NULL,0, raceloop, ret, 0, NULL); //Start first thread with
function raceloop
        hThread[1] = CreateThread(NULL,0, updateTime, ret, 0, NULL); //Start second
thread with function updateTime

        WaitForMultipleObjects( 2, hThread , true , INFINITE ); //Wait for both threads to
be finished

        //hThread[1] = CreateThread(NULL,0, updateTime, ret, 0, NULL); // updating UI a
last time

        //WaitForMultipleObjects( 1, hThread , true , INFINITE );

        CloseHandle(hThread[0]);
        CloseHandle(hThread[1]);

    }
    printf("\n\n\nRace has been finished");
    updateTime(ret); //Last time update for ranking
}

void gotoxy(int x, int y)
{

    HANDLE stdout;
    COORD pos;

    stdout = GetStdHandle(STD_OUTPUT_HANDLE);

    pos.X = x;
    pos.Y = y;

    SetConsoleCursorPosition(stdout, pos);
}

bool confirm() {

    char a;

    do{

        a=getch();
        if(a==89 || a==121) // Y and y , success
        {
            system("cls");
            return true;
        }

        if(a==78 || a==110) // N and n , failure
        {
            system("cls");
            return false;
        }
    }while(1); //As long as somebody presses y or n
}

```

```

void initUI(RACE *ret) {

    char a;
    int timeInt;
    char rounds[4];
    bool decision=false, test;
    a=0;

    do{
        //system("cls");
        printf("\nWelcome to the BG Carrera Application \n");
        printf("_____ \n\n\n");
        printf("\nWhat Game Mode you want to play?\n\n");
        printf("    1) Race\n");
        printf("    2) Knock Out\n");
        printf("    3) Time Attack\n\n");

        a=getch();
        test=false;

        if(a==49 || a==33)
        {
            printf("\n You want to choose Race? (y/n)");
            test=confirm();
            if( (test) )
            {
                decision=true;
                ret->match_Active=true;
            }
        }

        if(a==50 || a==34)
        {
            printf("\n You want to choose Knock Out? (y/n)");
            test=confirm();
            if( (test) )
            {
                decision=true;
                ret->maxRounds=4;
                ret->knockOut_Active=true;
            }
        }

        if(a==51 || a==35 || a==11)
        {
            printf("\n You want to choose Time Attack? (y/n)");
            test=confirm();
            if( (test) )
            {
                decision=true;
                ret->maxRounds=20;
                ret->timeAttack_Active=true;
            }
        }

    }while(!decision && !test); //Cant skip

    if(ret->match_Active)
    {
        do{
            system("cls");
            printf("\nWelcome to the BG Carrera Application \n");

```



```

printf("_____ \n\n\n");
printf("\nHow many rounds? (1-99)\n\n");

test=false;

fflush(stdin);
fgets(rounds, 4, stdin);
ret->maxRounds = atoi(rounds);
if(ret->maxRounds>0 && ret->maxRounds<100)
{
    printf("%d Rounds? (y/n)",ret->maxRounds);
    test=confirm();
}
else
    printf("\nYou have choosen %d. Please insert valid number (1-
99)",ret->maxRounds);

}while(!test);
}
if(ret->timeAttack_Active)
{
    do{
        system("cls");
        printf("\nWelcome to the BG Carrera Application \n");
        printf("_____ \n\n\n");
        printf("\nHow long? (min) (1-9)\n\n");

        test=false;

        fflush(stdin);
        fgets(rounds, 3, stdin);
        timeInt = atoi(rounds);
        if(timeInt>0 && timeInt<9)
        {
            printf("%d Minutes? (y/n)",timeInt);
            test=confirm();
            ret->maxTime = timeInt*60*CLOCKS_PER_SEC;

        }
        else
            printf("\nYou have choosen %d. Please insert valid number (1-
9)",ret->maxRounds);

    }.while(!test);
}

decision=false;

do{
    system("cls");
    printf("\nWelcome to the BG Carrera Application \n");
    printf("_____ \n\n\n");
    printf("\nHow many players want to participate? (1-4)\n\n");

    a=getch();
    test=false;

    if(a==49 || a==33)
    {
        printf("\n You want to choose 1 Player? (y/n)");
        ret->numberOfPlayers=1;
        test=confirm();
    }
}

```

```

        if( (test) )
            decision=true;
    }
    if(a==50 || a==34)
    {
        printf("\n You want to choose 2 Player? (y/n)");
        ret->numberOfPlayers=2;
        test=confirm();
        if( (test) )
            decision=true;
    }
    if(a==51 || a==35 || a==11)
    {
        printf("\n You want to choose 3 Player? (y/n)");
        ret->numberOfPlayers=3;
        test=confirm();
        if( (test) )
            decision=true;
    }
    if(a==52 || a==36)
    {
        printf("\n You want to choose 4 Player? (y/n)");
        ret->numberOfPlayers=4;
        test=confirm();
        if( (test) )
            decision=true;
    }
}while(!decision);

decision=false;

do{
    system("cls");
    printf("\nWelcome to the BG Carrera Application \n");
    printf("_____ \n\n\n");
    printf("Insert the name of the Player(s)\n\n");

    if(ret->numberOfPlayers>0)printf(" 1) %s\n",ret->players[0].playername);
    if(ret->numberOfPlayers>1)printf(" 2) %s\n",ret->players[1].playername);
    if(ret->numberOfPlayers>2)printf(" 3) %s\n",ret->players[2].playername);
    if(ret->numberOfPlayers>3)printf(" 4) %s\n",ret->players[3].playername);
    printf("\n 5) Continue\n");

    a=getch();
    test=false;

    if(a==49 || a==33)
    {
        printf("\n You want to rename %s? (y/n)",ret->players[0].playername);
        test=confirm();
        if( (test) )
        {
            getName(ret->players[0].playername);
        }
    }

    if((a==50 || a==34) && ret->numberOfPlayers>1)
    {
        printf("\n You want to rename %s? (y/n)",ret->players[1].playername);
        test=confirm();
        if( (test) )

```

```

        {
            getName(ret->players[1].playername);
        }
    }
    if((a==51 || a==35 || a==11) && ret->numberOfPlayers>2)
    {
        printf("\n You want to rename %s? (y/n)",ret->players[2].playername);
        test=confirm();
        if( (test) )
        {
            getName(ret->players[2].playername);
        }
    }
    if((a==52 || a==36) && ret->numberOfPlayers>3)
    {
        printf("\n You want to rename %s? (y/n)",ret->players[3].playername);
        test=confirm();
        if( (test) )
        {
            getName(ret->players[3].playername);
        }
    }
    if(a==53 || a==37)
    {
        printf("\n You want to continue? (y/n)");
        test=confirm();
        if( (test) )
            decision=true;
    }
}while(!decision);

decision=false;

do{
    system("cls");
    printf("\nWelcome to the BG Carrera Application \n");
    printf("_____ \n\n\n");
    printf("\nPrepare for the start\n\n");
    printf(" 1)We are ready!!!");

    a=getch();
    test=false;

    if(a==49 || a==33)
    {
        printf("\n You want to start the race? (y/n)");
        test=confirm();
        if( (test) )
            decision=true;
    }
}while(!decision);

buildTable(ret);

}

void getName(char name[20]) {

    char a;
    size_t ln;

    printf("\nWelcome to the BG Carrera Application \n");

```

```

printf("_____ \n\n\n");
printf("\nPlease insert the name\n\n");

fflush(stdin);
fgets(name, 20, stdin); //Max length of name: 20 characters

ln = strlen(name) - 1;
if (name[ln] == '\n') // if name is complete
    name[ln] = '\0'; // sign of a finished string
printf("Your name is '%s' ",name);

a = getch();
}

void buildTable (RACE *ret)
{
    int i;

    system("cls");

    gotoxy(0,1);
    printf("      I      I      I      I      ");

    for(i=0;i<ret->numberOfPlayers;i++) //Playernames
    {
        gotoxy(10+i*17,1);
        printf("%s",ret->players[i].playername);
    }

    gotoxy(0,2);
    printf("-----I-----I-----I-----I-----");

    for(i=0;i<ret->maxRounds;i++)
    {
        gotoxy(0,i+3);
        printf("Lap %02d I      I      I      I      \n",i+1);
    }
    gotoxy(0,ret->maxRounds+3);
    printf("-----I-----I-----I-----I-----");
    gotoxy(0,ret->maxRounds+4);
    printf("Total  I      I      I      I\n");
    gotoxy(0,ret->maxRounds+5);
    printf("Place  I      I      I      I\n");
}

DWORD WINAPI updateTime(LPVOID data)
{
    int i;

    RACE *ret;
    ret = (int) data;    // pointer is correct, ignore warning

    do{

        for(i=0;i<ret->numberOfPlayers;i++)
        {
            if(!ret->players[i].finished) //times are updated as long as the player did not
            finished the game
            {
                gotoxy(10+i*17,ret->players[i].rounds+3);
                if(!(ret->players[i].rounds== -1))

```

```

        printTime(ret,i,false); //Print the time of the players
current round

        gotoxy(10+i*17,ret->maxRounds+4);
        printTime(ret,i,true); //Update overall time
    }
    if(ret->players[i].finished) //If finished
    {
        gotoxy(10+i*17,ret->maxRounds+5);
        printf("    %d.",ret->players[i].rank); //print rank
    }
}
}while( ! (ret->finished) ); //As long as the game is running

return 0;
}

void printTime(RACE *ret, int player, bool total)
{
    int minuten, sekunden, millisekunden;
    clock_t timerX;
    timerX=clock();

    if(total) // prints the overall time
    {
        millisekunden = (float) (timerX-ret->currentTime);
        if(ret->players[player].finished)
            millisekunden = (float) ret->players[player].roundTime[ret-
>players[player].rounds]-ret->players[player].roundTime[0]; //Last round time minus the time of
when the race started
    }
    else{ //prints the current round time
        millisekunden = (float) (timerX-ret->players[player].roundTime[ret-
>players[player].rounds]); //current time minus the time of when the lap started
    }

    //Some crazy math nobody understands so don't even try it, you will fail anyway ;- )
    sekunden = millisekunden / 1000;
    minuten = sekunden / 60;
    millisekunden = millisekunden - 1000*sekunden;
    sekunden = sekunden - 60*minuten;
    printf("%02d:%02d:%03d\n",minuten,sekunden,millisekunden); //print
}

DWORD WINAPI raceloop(LPVOID data)
{
    int i,j,k;
    clock_t timer1;

    RACE *ret;
    ret = (int) data; // pointer is correct, ignore warning

    do{

        for(k=0; k < ret->numberOfPlayers ;k++)
        {
            playerCrossesLine(& ret->device, k); //checks if there are currently player
crossing the start point

            if(ret->device.activeTrackSensor[k] && !ret->activeSensor[k] && !ret-
>players[k].finished) //if player is currently crossing the start
            {

```

```

        ret->playerLine=k;
        ret->activeSensor[k]=true; //This is necessary to make sure that
the player just gets one new round at a time

        //Big issue if there are 2 active game modes

        if(ret->match_Active)
            match(ret);
        if(ret->knockOut_Active)
            knockOut( ret);
        if(ret->timeAttack_Active)
        {
            nextRound(&ret->players[ret->playerLine],ret-
>currentTime);

            /* @TO-DO: This is important for a future implementation
of a dynamic round cap, not yet working, especially with multithreading
            if(ret->players[ret->playerLine].rounds==ret-
>maxRounds-1)
            {
                //ret->maxRounds++;
                //buildTable(ret);
            }
            */
        }
        if( (!ret->device.activeTrackSensor[k]) && ret->activeSensor[k]) //TThis is
necessary to make sure that the player just gets one new round at a time
            ret->activeSensor[k]=false;

        timer1=clock();
        if( (timer1-ret->currentTime) > ret->maxTime && ret->timeAttack_Active)
//For TimeAttack, if the maximum time is reached
        {
            //Turn power off for all players
            ret->finished=true; //race has ended

            for(i=0;i<ret->numberOfPlayers;i++)
            {
                setPower( &ret->device,i,false ); //Turn off all tracks
                ret->players[i].finished=true;
                ret->players[i].endTime=clock();
                for(j=ret->players[i].rounds;j<ret->maxRounds+1;j++)
//Sets all rounds to endTime (important for export)
                    ret->players[i].roundTime[j]=ret-
>players[i].endTime=clock();
            }

            placingTimeAttack(ret); //Afterwards setting the ranks
        }
        updateTime(ret); //Updateting the UI
    }
}while( !(ret->finished) ); //As long as the game is running

return 0;
}

void countdown(RACE *ret) {

    int i=0;
    clock_t timer1;

    timer1=clock();

```

```

        while(i<4) { //As long as 3 seconds (for every step-1 (4-1 LEDs) one second )

            ret->currentTime= clock();

            if(! ((ret->currentTime-timer1)/CLOCKS_PER_SEC < 1) ) //If one second has
passed
            {
                timer1=clock(); //Reset CurrentTime
                ret->device.trafficLightStatus=i; //set current status

                setLights(& ret->device); //Setting the LEDs of the traffic lights
                i++;
                if(i==3) //after 3 seconds the game has officially started
                    ret->started=true;
            }
        }
    }
}

```

===== modes.h =====

```

#include "device.h"
#include "player.h"
#include <string.h>
#include <Windows.h>

```

```

struct RACE{

    bool finished;
    bool started;
    PLAYER players[4];
    DEVICE device;
    int maxRounds;
    bool match_Active;
    bool knockOut_Active;
    bool timeAttack_Active;
    clock_t maxTime;
    clock_t currentTime;
    int knockOutPlayer;
    int knockOutPlayerX[4];
    int playerLine;
    int numberOfPlayers;
    bool activeSensor[4];
};

```

```

typedef struct RACE RACE;

```

```

void initRACE(RACE *ret);
/*

```

The function initiates a RACE struct and opens a tcp connction to the UE9

```

*/

```

```

void match (RACE *ret);
/*

```

The function is the logic for the Game Mode race, it checks for specific conditions

```

*/

```

```

void knockOut (RACE *ret);

```



```

/*

The function is the logic for the Game Mode knockout, it checks for specific conditions

*/

void placingTimeAttack(RACE ret);
/*

The function is called after a game of 'Time Attack' has been finished.
It sets the ranking for all players after the number of finished rounds

*/

void run (RACE *ret);
/*

The function is the main function for a race.
In the beginning the countdown is started and
afterwards this function starts the threads for
    1)recieving and computing the race data
    2)updating the UI

afterwards
*/

void gotoxy(int x, int y);
/*

Generic Win32-API call for setting the Cursor in a CMD
X - Position in line
Y - Line

*/

bool confirm();
/*

The function is waiting for the users keyboard input.
If the user types 'y' or 'Y' it will return true
If the user types 'n' or 'N' it will return false

*/

void initUI(RACE *ret);
/*

The function initiates the UI as well as asking the user for information:
Game Mode
Number of Players
Players Name
Number of Rounds
Max. Time

*/

void getName(char name[42]);
/*

The function is asking for a players name

*/

```

```
void buildTable(RACE *ret);
/*
```

The function is dynamically building a time table

```
*/
```

```
DWORD WINAPI updateTime(LPVOID data);
/*
```

The function is updating the times in the time table  
returns 0

```
*/
```

```
void printTime(RACE *ret, int player, bool total);
/*
```

The function is printing the time of the current round or for the total time for specific player

player - The number of the player

total -

1) if true it prints the total time

2) if false it prints the current round time

```
*/
```

```
DWORD WINAPI raceloop(LPVOID data);
/*
```

The function is receiving the sensor status for every single player and afterwards computes them based on the Game Mode  
returns 0

```
*/
```

```
void countdown(RACE *ret);
```

```
/*
```

The function is starting the countdown for the race

```
*/
```

```
===== player.c =====
```

```
#include "player.h"
```

```
void initPLAYER ( PLAYER *start) { //initiation of the player struct, necessary in any means
```

```
    int i;
    start->rank=0;
    start->track=0;
    start->device=0;
    start->rounds=-1;
    start->bestRound=0;
    start->banned=false;
    start->finished=false;
    for(i=0;i<99;i++)
    {
        start->roundTime[i]=0;
    }
}
```

```

}

void nextRound(PPLAYER *start, clock_t currentTime) {

    // ----- Setting dedicated lap times -----
    int i=0;
    clock_t timer1;

    //This is due how it works, at the beginning, before the game starts, every player has a round
count of -1
    if(start->rounds>-1) //if player is crossing the line
    {
        timer1=clock();

        start->roundTime[start->rounds] = timer1-currentTime; //Set the total time of the
finished round

        // ----- Checking for best laps-----
        if(start->bestRound==0)
            start->bestRound=0;
        if( start->roundTime[start->bestRound] > start->roundTime[start->rounds] )
            start->bestRound=start->rounds;

        //Initiate new round
        start->rounds++;
        start->roundTime[start->rounds]=clock();
        start->roundTime[start->rounds+1]=clock(); //Important for export

    }
    else
    {
        //Start the game, initiation
        start->rounds++;
        start->startTime=clock();
        start->roundTime[start->rounds+1]=clock();
    }
}

```

===== player.h =====

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "stdbool.h"
#include <string.h>

typedef struct {

    int rank;
    int track;
    int device;
    int rounds;
    int bestRound;
    char playername[20];
    bool banned;
    bool finished;
    clock_t roundTime[100];
    clock_t startTime;
    clock_t endTime;

}PLAYER;

```

```
void initPLAYER ( PLAYER *start);
/*
```

The function initiates the struct PLAYER

```
*/
void nextRound(PLAYER *start, clock_t currentTime);
/*
```

The function initiates a new round for the specific player  
\*start - the specific player

```
*/
```

```
===== device.c =====
```

```
#include "device.h"
```

```
void initDEVICE(DEVICE *ret) {
    int i=0;
    startConnectionUE9(&ret->ue9);

    for(i=0;i<4;i++)
    {
        ret->activeTrack[i]=false;
        ret->activeTrackSensor[i]=false;
    }
    ret->trafficLightStatus=4;
    setLights(ret);
    setPower( ret,0,false );
    setPower( ret,1,false );
    setPower( ret,2,false );
    setPower( ret,3,false );

}

void setLights( DEVICE *ret) {

    if(ret->trafficLightStatus==0) //First red LED
    {
        setPortUE9(&ret->ue9, 0, 1);
    }
    if(ret->trafficLightStatus==1) //Second red LED
    {
        setPortUE9(&ret->ue9, 1, 1);
    }
    if(ret->trafficLightStatus==2) //Third red LED
    {
        setPortUE9(&ret->ue9, 2, 1);
    }
    if(ret->trafficLightStatus==3) //All red LED off, green LED on
    {
        setPortUE9(&ret->ue9, 0, 0);
        setPortUE9(&ret->ue9, 1, 0);
        setPortUE9(&ret->ue9, 2, 0);
        setPortUE9(&ret->ue9, 3, 1);
    }
    if(ret->trafficLightStatus==4) //ALL Lights OFF, it's getting dark
    {
        setPortUE9(&ret->ue9, 0, 0);
    }
}
```

```

        setPortUE9(&ret->ue9, 1, 0);
        setPortUE9(&ret->ue9, 2, 0);
        setPortUE9(&ret->ue9, 3, 0);
    }
}

void playerCrossesLine(DEVICE *ret, int track) {

    // ----- Check Port for Digital High -----
    int i=-1;
    getPortUE9(&ret->ue9, track+8, &i); //Gets port

    if(i>0) //If active
        ret->activeTrackSensor[track]=true;
    else //If inactive
        ret->activeTrackSensor[track]=false;
}

void setPower(DEVICE *ret, int track, bool energy ) {

    if(energy) //power track on
    {
        ret->activeTrack[track] = 1;
        setPortUE9(&ret->ue9, (track+4+8), 1); // see comments
    }

    if(!energy) //power track off
    {
        ret->activeTrack[track] = 0;
        setPortUE9(&ret->ue9, (track+4+8), 0); // see comments
    }
}

void startConnectionUE9 (CONNECTION *ret) {

    ret->ue9_port = DEFAULTPORT;

    //Opening TCP connection to UE9
    if( !( ret->socketFD = openTCPConnection(DEFAULTTIP, ret->ue9_port)) < 0) //returns true
    if succesfull, native UE9
        printf("Connected ");

    //Getting calibration information from UE9 (Necessary for communication)
    if( ! (getCalibrationInfo(ret->socketFD, &ret->caliInfo) < 0))
        printf("\nCalibrated \n");
}

void setPortUE9(CONNECTION *ret, int portNumber, int value) {

    /*
        0-7   FIO0-FIO7, 0-3   traffic lights
        8-11   Sensors
        12-15   tracks
        8-15   EIO0-EIO7

    */

    if((ret->error = eDO(ret->socketFD, portNumber, value)) != 0) // Reads port, return true if
    succesfull, native UE9
        printf("setPort unsuccesfull");
}

```

```

void getPortUE9(CONNECTION *ret, int portNumber, int *value) {

    /*
        0-7   FIO0-FIO7,      0-3   traffic lights
        8-11   Sensors
        12-15   tracks
        8-15   EIO0-EIO7

    */

    if((ret->error = eDI(ret->socketFD, portNumber, &ret->lngState)) != 0) // Set port, returns
true if succesfull, native UE9
        printf("Could not read the state of Port %d", portNumber);

    *value = (int) ret->lngState;

}

void closeConnectionUE9 (CONNECTION *ret) {

    if(ret->error > 0) //Connection issue
        printf("Received an error code of %ld\n", ret->error);
    if(closeTCPConnection(ret->socketFD) < 0) // closes connction, returns true if succesfull, native
UE9
    {
        printf("Error: failed to close socket\n");
    }

}

===== device.h =====

#include "ue9.h"
#include "stdbool.h"
#include "definitions.h"

typedef struct {
    int socketFD, i;
    ue9CalibrationInfo caliInfo;
    long error;
    long lngState;
    int ue9_port;

}CONNECTION;

typedef struct {

    bool activeTrack[4];
    bool activeTrackSensor[4];
    int trafficLightStatus;
    CONNECTION ue9;

}DEVICE;

void initDEVICE(DEVICE *ret);
/*

```

The function initiates the Device Struct

\*/

```
void setLights( DEVICE *ret);  
/*
```

The function sets the LEDs of the traffic light

\*/

```
void playerCrossesLine(DEVICE *ret, int track);  
/*
```

The function checks the status of a specific track  
track - Number of the player/track

Writes result in struct DEVICE.activeTrackSensor[track]

\*/

```
void setPower(DEVICE *ret, int track, bool energy );  
/*
```

The function is setting the power off or on on a specific track  
track - Number of the player/track  
energy -

- 1) if true it turns the power on
- 2) if false it turns the power off

\*/

```
void startConnectionUE9 (CONNECTION *ret);  
/*
```

The function initiates the connection to the UE9

\*/

```
void setPortUE9(CONNECTION *ret, int portNumber, int value);  
/*
```

The function sets one port of the UE9  
portNumber - Number of the specific port  
value - 1 for on, 0 for off

\*/

```
void getPortUE9(CONNECTION *ret, int portNumber, int *value);  
/*
```

The function receives the status of one port  
portNumber - Number of the specific port  
value - int pointer for writing the status, 1 for on, 0 for off

\*/

```
void closeConnectionUE9 (CONNECTION *ret);  
/*
```

The function stops the connection to the UE9

\*/



=====

Sourcecode von ue9.c wurde aus Platzgründen nicht eingefügt.

Siehe dafür LabJacks Support Webseite

Der gesamte Quellcode steht unter <https://github.com/schattenan/EschwegeCarreraProjectInC>

## 10.2. Drucken einer Urkunde

|    | A         | B         | C         | D         | E |                   |
|----|-----------|-----------|-----------|-----------|---|-------------------|
| 1  | Race      |           |           |           |   | //Game Mode       |
| 2  |           | 4         |           |           |   | //NumberOfPlayers |
| 3  |           | 5         |           |           |   | //NumberOfRounds  |
| 4  | Player1   | Player2   | Player3   | Player4   |   | //PlayerName      |
| 5  | 00:37:256 | 00:37:256 | 00:37:256 | 00:37:256 |   | //Round0          |
| 6  | 00:37:255 | 00:37:256 | 00:37:256 | 00:37:256 |   | //Round1          |
| 7  | 00:37:256 | 00:37:255 | 00:37:256 | 00:37:256 |   | //Round2          |
| 8  | 00:37:256 | 00:37:256 | 00:37:256 | 00:37:256 |   | //Round3          |
| 9  | 00:37:256 | 00:37:256 | 00:37:256 | 00:37:256 |   | //Round ...       |
| 10 | 1.        | 2.        | 3.        | 4.        |   | //Ranking         |
| 11 |           | 2         | 3         | 4         | 1 | //BestRound       |
| 12 | 02:33:538 | 02:33:538 | 02:33:538 | 02:33:538 |   | //TotalTime       |
| 13 | End       |           |           |           |   | //Signal für Ende |
| 14 |           |           |           |           |   |                   |

### 10.3. Struktogramm

