INFOSAMMLUNG

[Version 1.8]

Herbert MOLDAN, 0728098
Michael PRÄAUER, 0728095

Inhaltsverzeichnis

Projektaufbau

Generelles:

Eye-Tracker

- es handelt sich hierbei um einen Tobii Eye Tracker x50
- der Eye-Tracker ist an der Fakultät für Informatik am Computer-Grafik Institut, Favoritenstraße 9-11, verfügbar. Ansprechpartner und Verantwortlicher für den Eye-Tracker ist Herr Prof. Michael Wimmer

Tobii Eye Tracker 1
(http://web.unife.it/progetti/neurolab/image/Tobiix50.jpg)

Körperhaltung

Die besten Ergebnisse erzielten wir mit ca. 50cm Abstand vom Bildschirm.
Wichtig ist darauf zu achten Kopf und Körper bei der Kalibrierung und während dem Tracking möglichst wenig zu bewegen.

Probleme

Nach längerem testen kristallisierten sich folgende Probleme heraus:

- Träger harter Kontaktlinsen können den Eye Tracker nicht verwenden. Bei weichen Kontaktlinsen ergaben sich keine Probleme.
- Brillenträger können den Eye Tracker –zumindest mit Brille- nicht verwenden.
- Die „Streuung“ des Eye Trackers beträgt bis zu 2cm. (Abstand von „vom Eye Tracker gemessenen Punkt“ zu „wirklichem Punkt“)
- Durch den relativ langen Kalibrierungsvorgang und dadurch dass der Eye Tracker relativ sensibel auf Positionsveränderungen des Körpers reagiert ist er leider nicht geeignet für „spontane Interaktionen“ wie bei Kunstausstellungen und ähnlichem.

Vorbereitung:

Eye-Tracker

Der Eye-Tracker wird unter dem Bildschirm aufgestellt und angesteckt.
Wichtig hierbei ist die Reihenfolge:

- zuerst die FirewireVerbindung
- danach die USB-Verbindung
- am Ende erst die Spannungsversorgung

Clear View 2:

Voreinstellungen

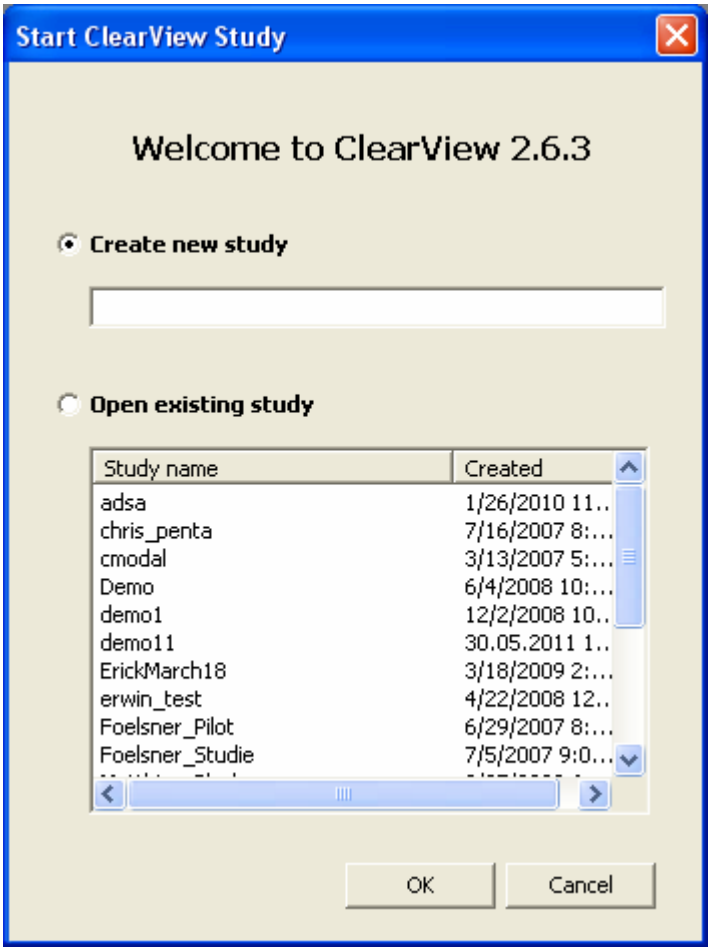
Um den Eye Tracker verwenden zu können muss der Eye Tracker vorab auf jeden Probanden eingestellt werden.

Hierzu wird die Software Clear-View 2 verwendet:

Clear View 2, Abb. 1

Neue Studie erstellen

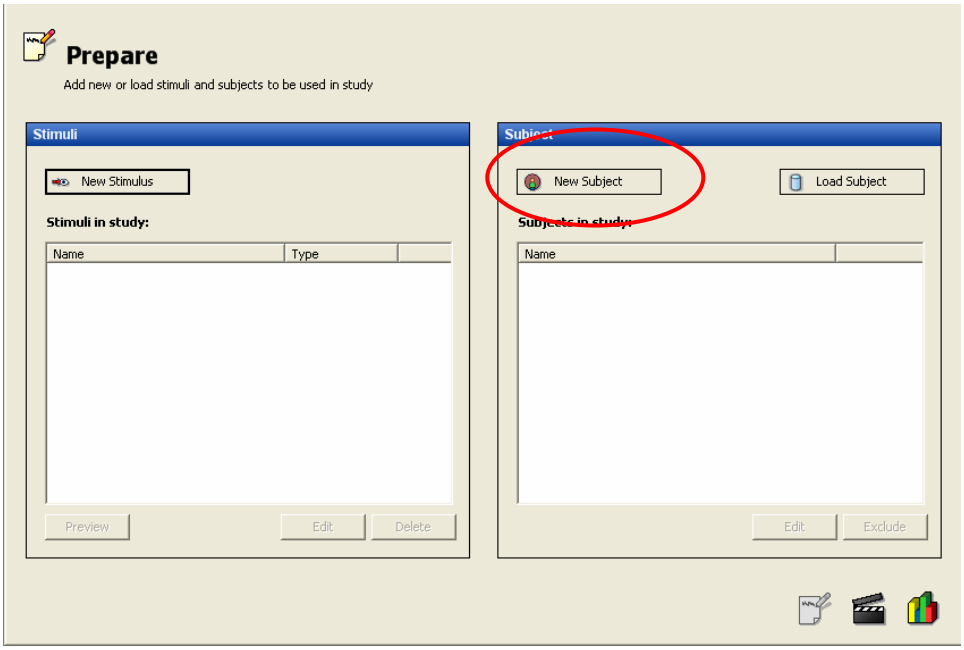
Hier der aktuellen Studie ein beliebiger Titel zugeordnet werden.
Zu jeder Studie können mehrere Probanden eingeteilt werden, zu jedem Probanden wiederum mehrere Eye-Tracker Konfigurationen.



Clear View 2, Abb. 2

Neues Subjekt erstellen

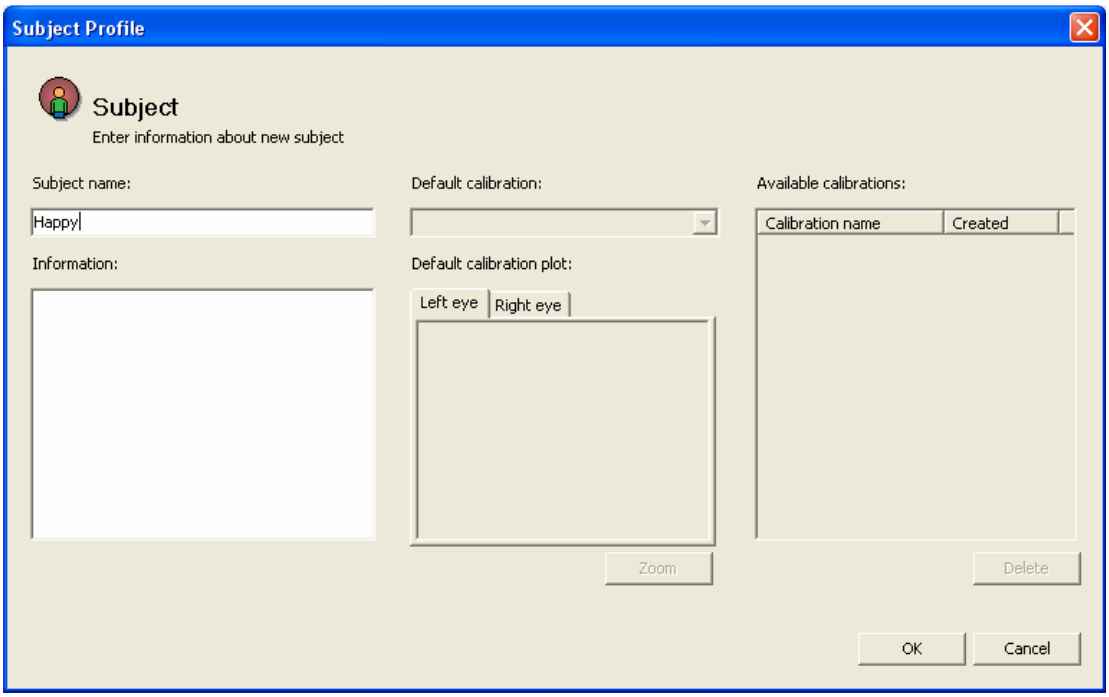
Danach kommt man auf die „Vorbereitungs-Seite“. Hier können die einzelnen Probanden mithilfe der Schaltfläche „new Subject“ hinzugefügt werden.



Clear View 2, Abb. 3

Probanden eintragen

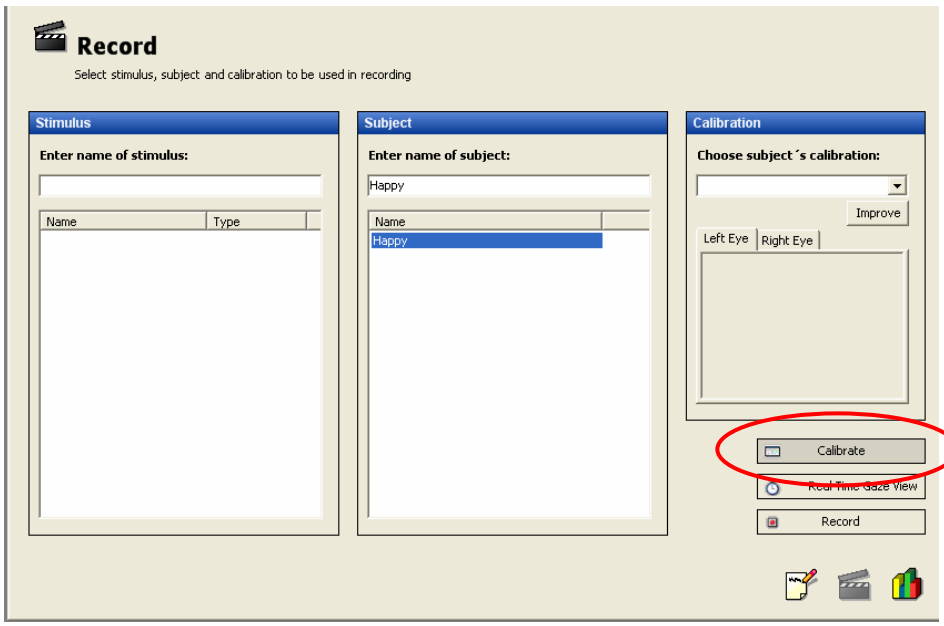
Danach werden die einzelnen Probanden eingetragen.
Die Empfohlene Vorgangsweise ist alle Teilnehmer vorab einzutragen, die Kalibrierung jedoch für jeden Probanden separat vorzunehmen sobald er vorm Eye-Tracker platzgenommen hat.



Clear View 2, Abb. 4

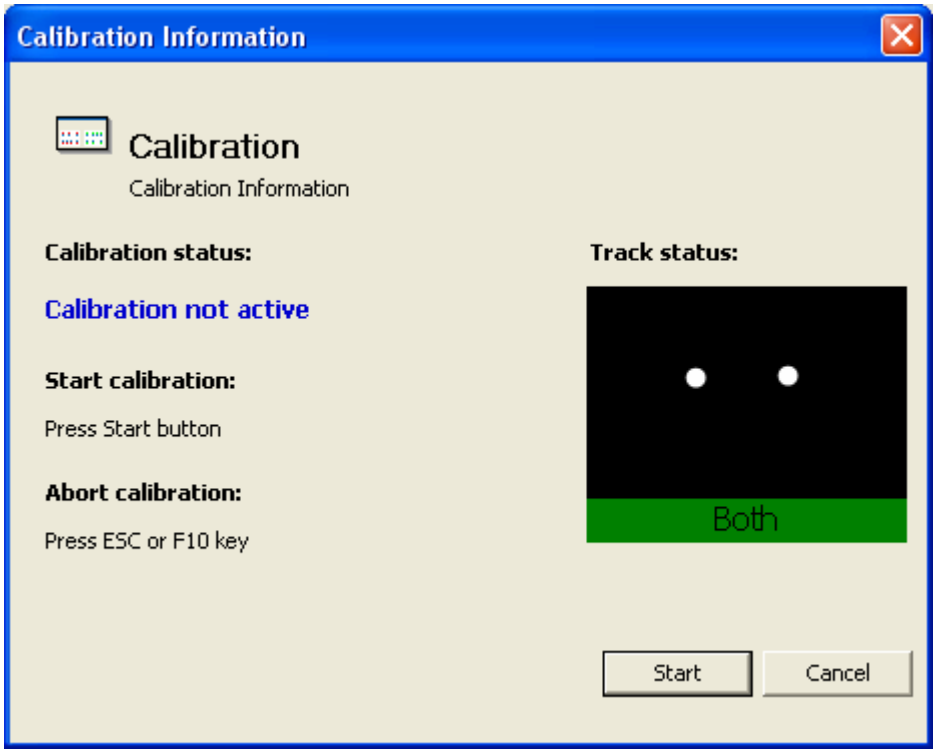
Kalibrierung

Um den Eye Tracker für eine Person (Subject) zu kalibrieren muss diese ausgewählt und danach die Kalibrierung über den Button „Calibrate“ gestartet werden.



Clear View 2, Abb. 5

Nun sucht der Eye Tracker das Augenpaar.
Erst wenn beide Augen sauber und permanent erkannt werden kann die Kalibrierung starten.
Die Augen erkennt man durch die beiden weißen Punkte im „Track Status“. Im Textfeld darunter wird die erfolgreiche Erkennung beider Augen mit dem Wort „Both“ angezeigt.



Clear View 2, Abb. 6

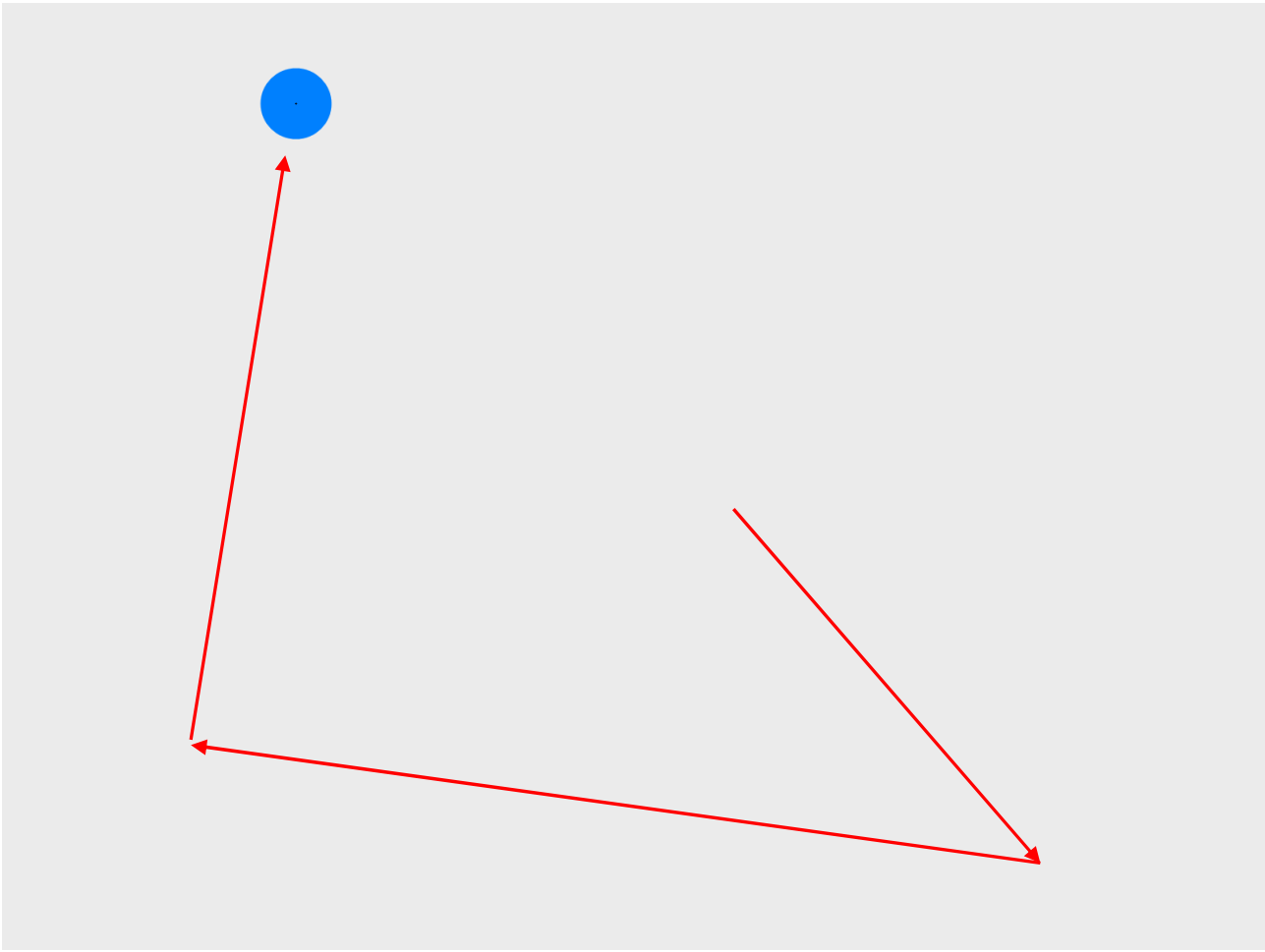
Hinweis: Wenn die Anzeige „Both“ blinkt bzw. die beiden weißen Punkte springen kann der Eye Tracker von diesem Kandidaten leider nicht verwendet werden.

Nun kann durch drücken auf den Button „Start“ der Kalibrierungsvorgang gestartet werden.

Hinweise:

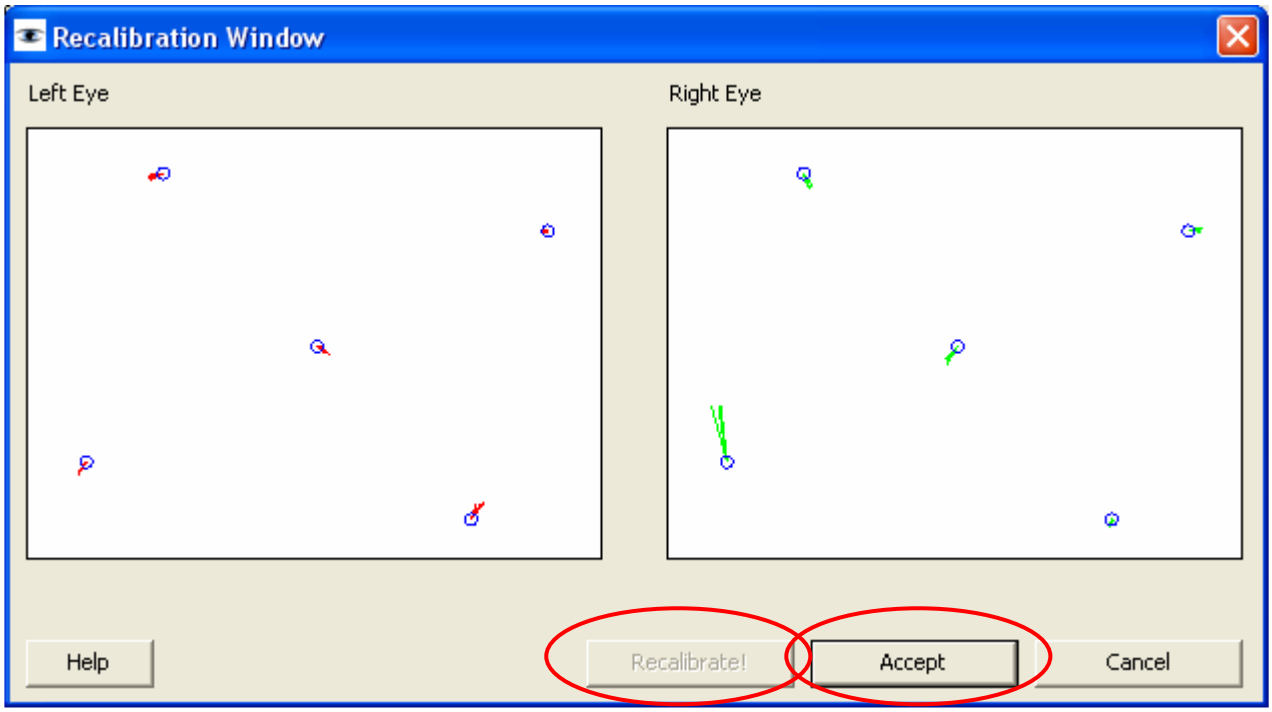
- Der Proband muss nun direkt und mittig vor dem Bildschirm bzw. vor dem Eye Tracker platziert sein.
- Beide Augen müssen einwandfrei erkannt werden
- Der Abstand zum Bildschirm sollte ca. 40-50cm betragen

Nun soll der Proband –möglichst ohne den Kopf zu bewegen- mit den Augen den blauen Punkten auf dem Bildschirm folgen.



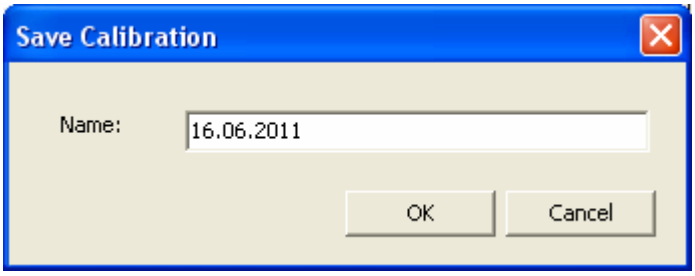
Clear View 2, Abb. 7

Nach Abschluss des Vorganges wird das Ergebnis der Kalibrierung angezeigt.
Wenn die Kalibrierung erfolgreich war ist der Button „Accept“ aktiv.
Sollte die Kalibrierung nicht bei allen Punkten erfolgreich gewesen sein wird die Option „Recalibrate“ angeboten.
Danach werden die entsprechenden Punkte erneut angezeigt und die Kalibrierung für diese Punkte wiederholt.



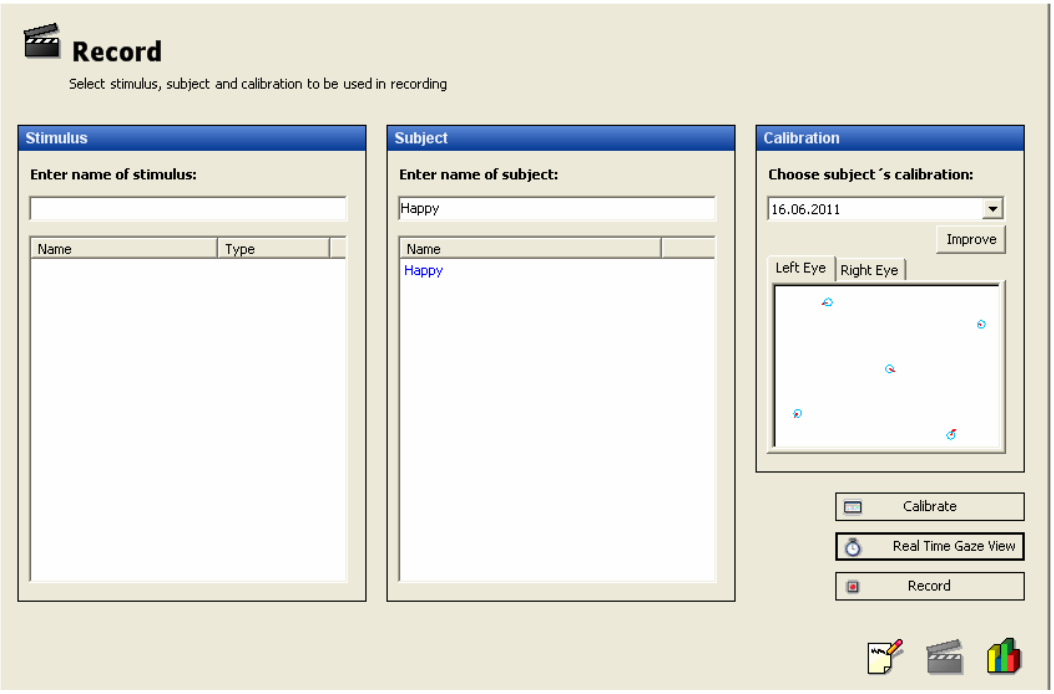
Clear View 2, Abb. 8

Die durchgeführte Kalibrierung wird danach zu dem vorhin ausgewählten Subjekt hinzugefügt.



Clear View 2, Abb. 9

Im Übersichtsfenster sieht man danach die Einstellungen.
Der Eye Tracker ist nun korrekt kalibriert und kann vom Probanden verwendet werden.



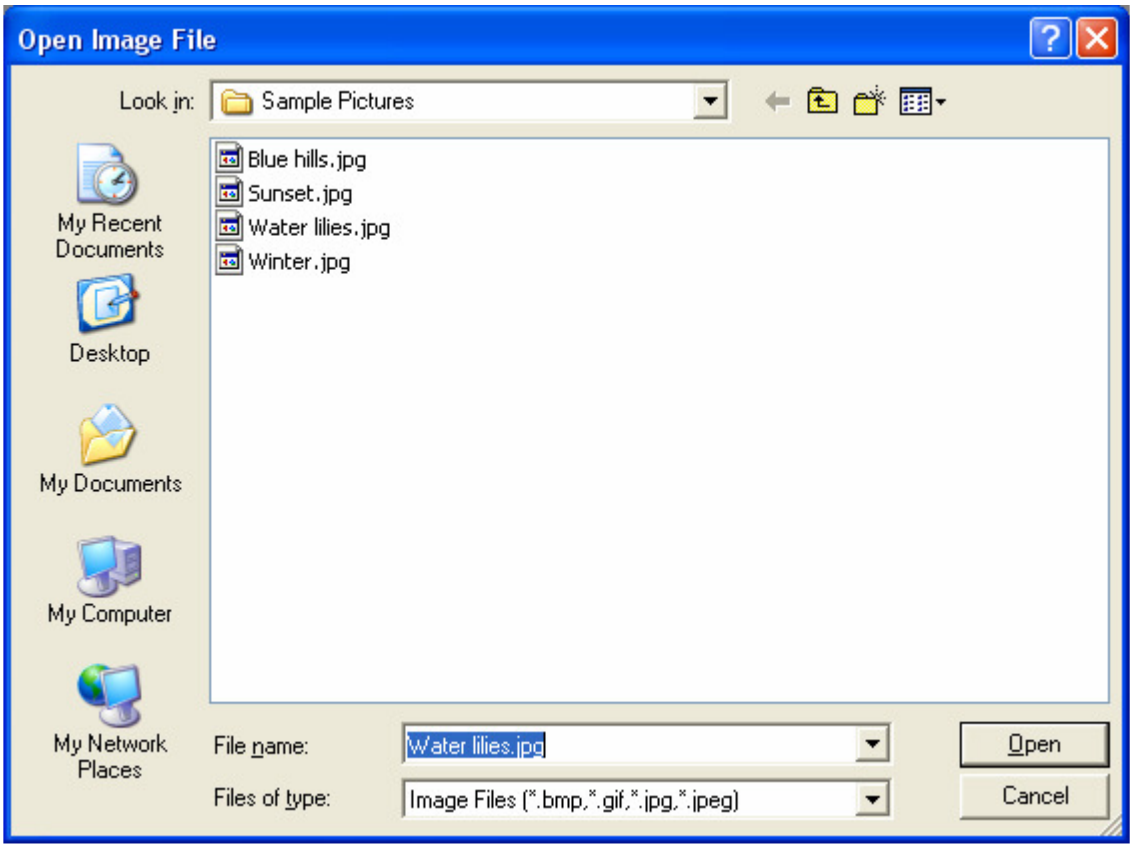
Clear View 2, Abb. 10

Eye Tracker testen / Stimuli

Clear View bietet die Möglichkeit die aktuellen Einstellungen des Eye Trackers zu testen indem man über den Button „Real Time Gaze View“ ein beliebiges Bild laden und parallel dazu den aktuellen Blickpunkt in Echtzeit einblenden lassen kann.

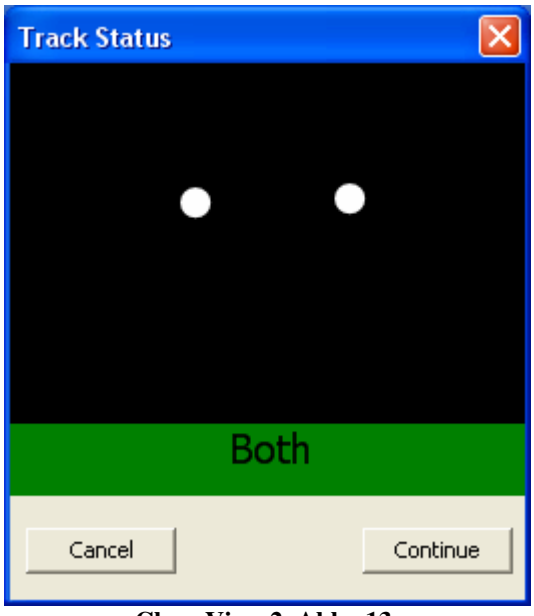
Clear View 2, Abb. 11

Es werden von Clear View zu diesem Zweck alle gängigen Bildformate unterstützt.



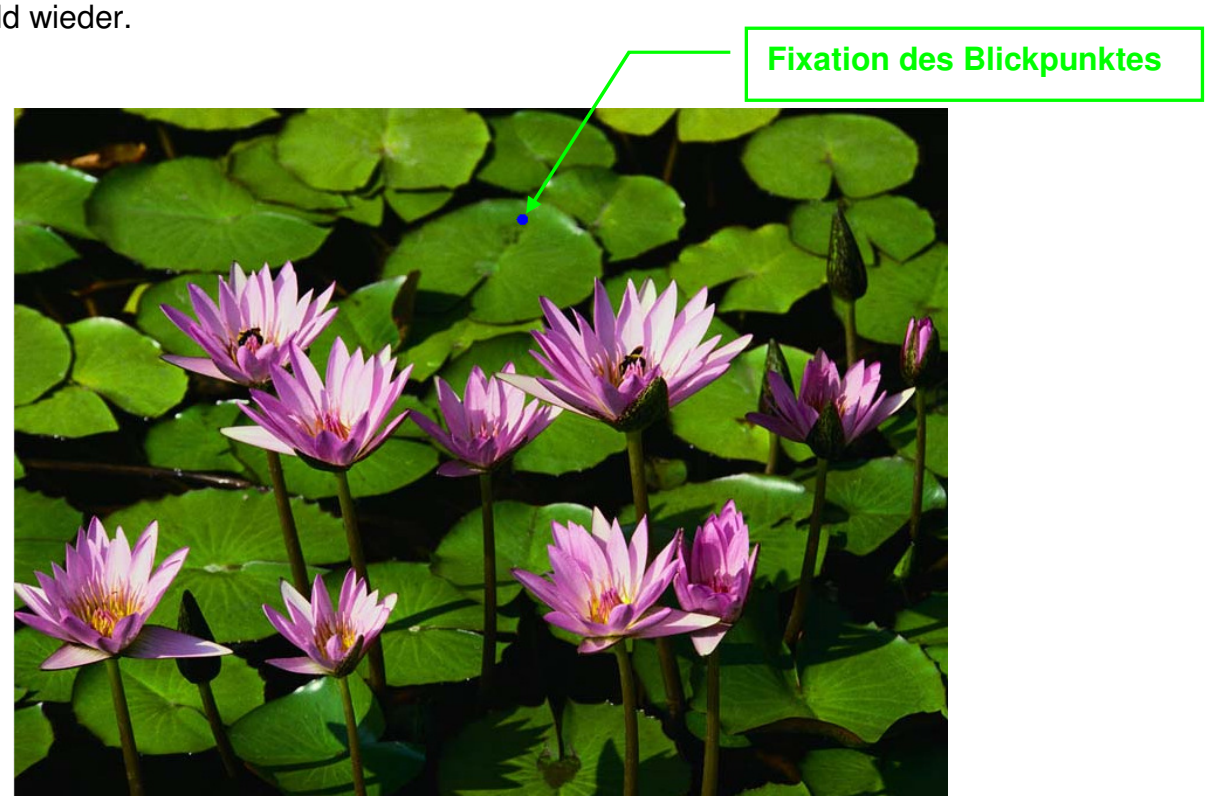
Clear View 2, Abb. 12

Jetzt zeigt der Eye Tracker an ob er das Augenpaar erkennen kann.
Mit einem Klick auf „Continue“ wird die Anzeige gestartet.



Clear View 2, Abb. 13

Nun wird das gewählte Bild am Bildschirm angezeigt.
Die aktuellen Augenbewegungen spiegeln sich in den Bewegungen des blauen Punktes auf dem Bild wieder.



Clear View 2, Abb. 14

Eye Tracker API

Zur grundsätzlichen Steuerung des EyeTrackers verwenden wir das vorgefertigte Programm „EyeLib“ welches uns bereits zur Verfügung gestellt wurde. Das Programm wurde jedoch von uns noch etwas modifiziert und für unsere Zwecke angepasst. EyeLib wurde dadurch relativ stark vereinfacht indem Funktionalitäten die wir für unsere Anwendung nicht benötigen, wie zum Beispiel das gleichzeitige steuern mehrerer EyeTracker oder die Steuerung über ClearView, entfernt wurden. Es wurden nur die Methoden zum überprüfen der Hardware, zum Speichern bzw. Laden einer Kalibrierung, zum initialisieren und zum starten und stoppen des Trackers belassen.
Neben der Steuerung des EyeTrackers erstellt das Programm ein „Gaze-File“ das alle Daten die der EyeTracker während der Sitzung lieferte beinhaltet.
Auch hier war eine kleine Modifikation nötig um die Ticks seit Systemstart ins Gaze-File aufzunehmen. Diese Ticks werden dann für die Synchronisation mit den Objekt-Daten des Spiels benötigt. Es werden dann also die Ticks im Gaze-File mit den Ticks im Object-File dass das Spiel liefert synchronisiert.
Weiters besteht mit EyeLib die Möglichkeit auch in Echtzeit während der Sitzung die Daten des EyeTrackers live auszulesen. Das wurde von uns vor allem für Debugging-Zwecke verwendet indem die aktuellen Gaze-Daten live auf die Konsole ausgegeben wurden.

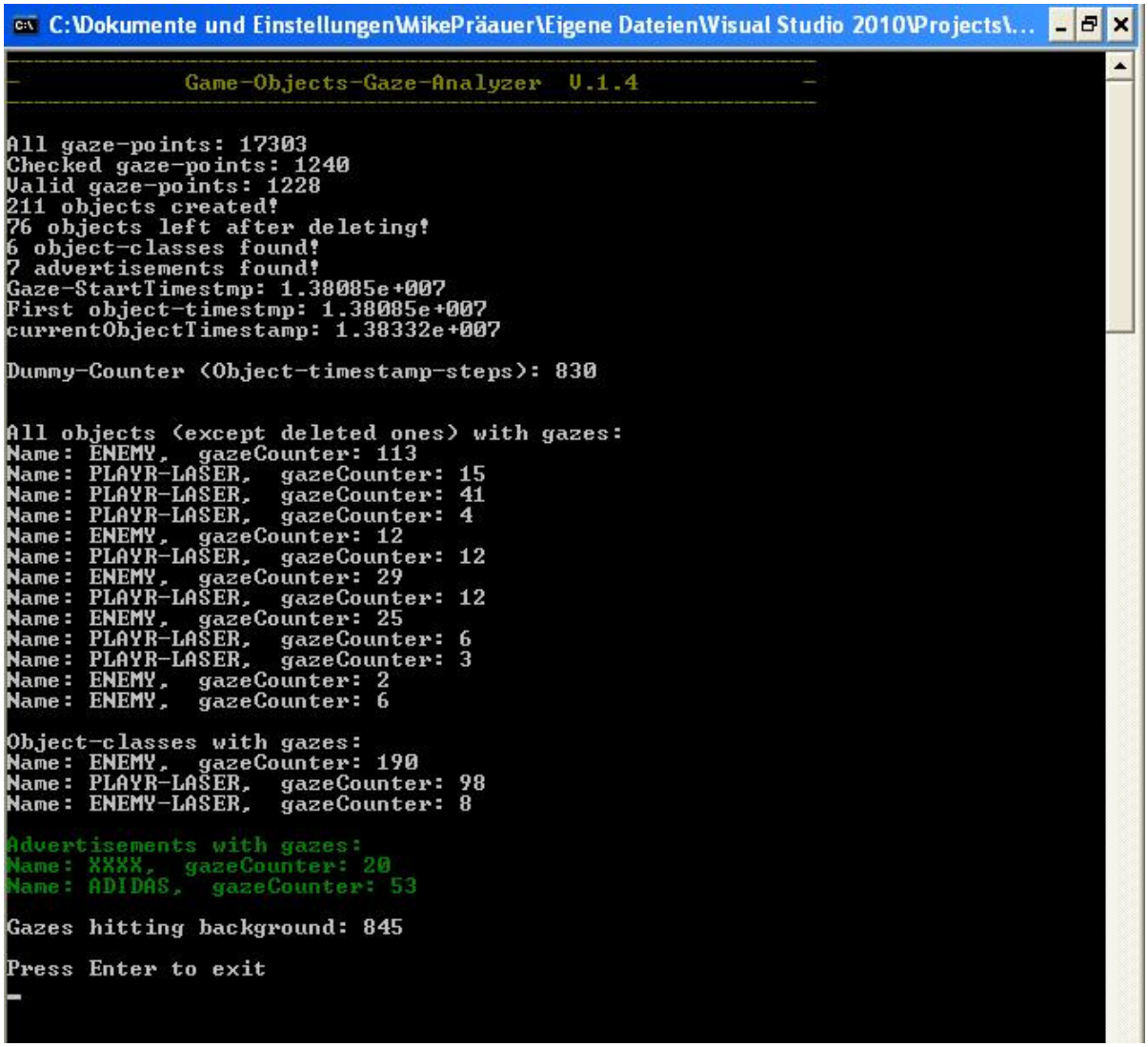
Mit den Daten des Gaze-Files und den Positions- und Objektdaten die die Torque-Engine aus dem Spiel speichert kann nun unsere Auswertung erstellt werden.
Das Programm „GazeLib“ welches uns ebenfalls zur Verfügung gestellt wurde beinhaltet bereits die Funktionalität zum einlesen von Gaze-Files. GazeLib wurde von uns nun so erweitert dass auch Object-Files eingelesen werden können. Weiters erfolgt die komplette Auswertung der Gaze- und Object-Daten über dieses Programm.
Das größte Problem dabei stellt die Synchronisation dar. Da ein zeitgleiches starten von EyeTracker und Spiel quasi unmöglich scheint sind wir den Weg gegangen die Timestamps bzw. Ticks seit Systemstart von EyeTracker und Spiel zu synchronisieren. Da das Spiel die Daten in Intervallen zu 32ms liefert und der EyeTracker ein Intervall von 20ms verwendet musste eine eigene Logik entwickelt werden um alle Daten abzugleichen.

Funktionalität

Nach dem Einlesen des Gaze-Files (*.gaz) wird in ähnlicher Weise unser Object-File Zeile für Zeile eingelesen. Das Object-File ist nach XML-Standard aufgebaut und enthält Informationen über sämtliche Objekte im Spiel und die jeweiligen Positionen zu jedem Timestamp. Neue Objekte werden mit dem Tag <newObject:xxxx> (xxxx steht für die Object-ID) gekennzeichnet und mit sämtlichen Objektdaten (ID, Name, Größe in x- und y-Richtung) in einer Liste gespeichert. Parallel dazu wird das Objekt in eine Liste von Objektklassen (PLAYER, ENEMY, LASER, …) eingetragen sofern diese Klasse bisher noch nicht vorgekommen ist.
Positionseinträge werden mit dem Tag <pos:xxxx> (xxxx steht für die Objekt-ID) gekennzeichnet und direkt verarbeitet. Dabei werden sämtliche Positionseinträge die zum gleichen Spiel-Timestamp gehören gesammelt und die angegebenen x- und y-Positionen den jeweiligen Objekten als momentane Position zugeordnet.
Über die aktuelle Position und den gespeicherten Größen der einzelnen Objekte können daraufhin die Objekt-Umschließenden Bounding-Boxes berechnet werden.

Daraufhin werden alle Gaze-Timestamps die im Bereich zwischen dem aktuellen und dem nächsten Spiel-Timestamp liegen gesammelt und direkt ausgewertet.

Zu jedem dieser Gaze-Timestamps werden nun die Blickpositionen des Spielers einzeln mit den aktuellen Positionen der Objekte und deren Bounding-Boxes im Spiel verglichen. Die Objekte werden in der Reihenfolge ihrer „Tiefe“ untersucht, also als erstes sämtliche Vordergrundobjekte wie PLAYER, ENEMY, HEALTH-OBJECT usw. Bei keiner Übereinstimmung (d.h. der Blick traf keines der Vordergrundobjekte) werden als nächstes die Werbungen im Hintergrund untersucht. Bei wieder keiner gefundenen Übereinstimmung traf der Blick nur den Hintergrund.
Bei Übereinstimmung von Timestamp, Blickposition und Objekt-Bounding-Box wird der gazeCounter des jeweiligen Objektes um 1 erhöht.
Nachdem alle Daten des Gaze- und des Object-Files ausgewertet wurden werden alle Gaze-Hits der jeweiligen Objekt-Klasse zugeordnet. Das Ergebnis (was während des Spiels wie oft angesehen wurde) wird schließlich auf die Konsole ausgegeben.
In der aktuellen Version 1.4 sieht eine Auswertung folgendermaßen aus:



Game-Objects-Gaze-Analyser, Abb. 1

Besonderheiten

Unsere Auswertung funktioniert komplett unabhängig von der eingestellten Bildschirmauflösung des Test-Rechners. Da die Gaze-Daten ohnehin immer zwischen den Werten [0,0] für das linke obere Eck des Bildschirmes und [1,1] für das rechte untere Eck

des Bildschirmes liegen haben wir auch die Positionen der Objekte im Spiel entsprechend angepasst.

Mithilfe der Formeln x[0,1] = (xGame + 67.5) / 135 und y[0,1] = (yGame + 35) / 70 erhalten wir Auflösungsunabhängig die gewünschten Objektpositionen zwischen 0 und 1.

Die ersten Versionen unserer GazeLib-Modifikation haben durch die sehr großen Datenmengen in Gaze- und Object-File relativ lange Rechenzeiten gefordert weshalb viel Zeit in die Optimierung der Funktionen investiert werden musste.
Die aktuelle Version benötigt jetzt nur mehr einen Bruchteil der Zeit der Anfänglichen Versionen für die Auswertung der Daten.

Sämtliche ungültige Gaze-Daten und Objekt-Positionsdaten werden, um weiter Rechenzeit einzusparen, bereits vor ihrer Verarbeitung herausgefiltert.

2Do / Infos

-Grafische Auswertung der Ergebnisse (Momentan werden die Ergebnisse nur auf der Konsole ausgegeben) z.B. Histogramm
-Implementierung unserer Funktionalität in die Applikation „Iyzer“
-Experimente mit unterschiedlichem Aufbau durchführen

InGame-Tracking

Um eine korrekte Auswertung der Aufmerksamskeitspunkte des Beobachters erhalten zu können ist es notwendig zu jedem Zeitpunkt zu wissen welches Objekt betrachtet wird. Hierzu war es erforderlich die Trajektorie jedes einzelnen Objektes aufzuzeichnen und persistent zu sichern.

Folgende Punkte waren hier primär zu beachten:
- die Auswertung sollte einheitliche Ausgaben erzeugen
- die Performance darf –auch nachdem das Spiel längere zeit ausgeführt wird- nicht beeinträchtigt werden
- die Daten müssen persistent gesichert werden
- die Daten müssen zu konkreten Zeitpunkten zuordenbar sein

Optional wäre es noch erstrebenswert die Eigenschaft bzw. den Tracker so zu Generalisieren das sie –ohne Adaptierungen- an jedes Objekt im Spiel angehängt werden kann.
Auch das definieren von Aufzeichnungsgrenzen –innerhalb welcher die Position des Objektes Aufgezeichnet wird- wäre von Vorteil.

Log-File

Um die Daten –auch für spätere Überprüfungen- persistent speichern zu können wird der Output-Stream in ein .log-File gesichert.
Die Größe des Log-Files beträgt ungefähr 1MB pro gespielte Minute.
(Mit Durchschnittlich 20.000-25.000 Einträgen pro Minute)
Der Trigger -welche die Aufzeichnungseinträge pro sichtbaren Objekt in den Stream schreibt- wird alle 32ms ausgelöst.

Einträge

Ein neues Objekt wird mit folgendem Eintrag angezeigt: (Attribut ist hier die eindeutige Objekt-ID)

```
<newObject:1356>1356</newObject:1356>
```

Danach wird –für ein leichteres Verständnis- der Name des Objekts angezeigt.
(Dieser ist frei definierbar, siehe „Tracker (Behavior)“)

```
<name:1356>PLAYER</name:1356>
```

Um für den Eye-Tracker auch die entsprechende „Bounding-Box“ berechnen zu können werd die Größe des Objektes mit ausgegeben. (inGame Größe)

```
<sizeX:1356>5.638</sizeX:1356>
<sizeY:1356>7.603</sizeY:1356>
```

Performance

Anfänglich wurde versucht pro Objekt einen Ausgabe-Stream zu erstellen.
Dies resultierte jedoch in unakzeptablen Performance-Einbrüchen.
Somit wurde entschieden sich in einem –bereits vorhandenen Ausgabe-Stream- einzuschalten und die Log-Einträge dort zu sichern.

Weiters konnte beobachtet werden dass jedes Objekt immer neu generiert wird. (Objekte welche den Canvas verlassen haben werden auch nicht freigegeben. Der Speicherplatz bleibt somit belegt.)
Als Resultat ergab sich dadurch ein untragbarer Performance-Einbruch nach ca. 1:30 Minuten Spielzeit.

Auch die Programmlogik welche verhindern sollte dass sich „nicht-sichtbare Objekte“ in den Stream eintragen könnte hier keine Verbesserung schaffen da das reine Aufrufen der „Update“-Methode jedes einzelnen (auch nicht sichtbaren Objektes) das System hier bereits an seine Grenzen stoßen ließ.
(Die Anzahl der Objekte steigt mit der Spielzeit linear mit an)

Hierzu war eine weitere Programmlogik erforderlich welche alle Objekte die einen Update-Call erhalten sollte korrekt verwaltet und sich um die entsprechende Zuordnung kümmert.

Timestamps

Es erfolgen laufend Einträge welche die Position des Objektes wiedergeben.
(Abhängig von den definierten World-Limits, siehe „Tracker (Behavior)“)
Hierbei handelt es sich um inGame Koordinaten, Format: „Position-X“; „Position-Y“; „Timestamp“

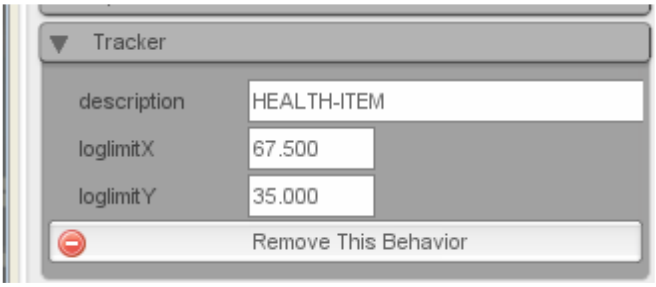
```
<pos:1356>-42.431; -2.109; 5063953;</pos:1356>
<pos:1361>52.5; -24.163; 5063953;</pos:1361>
<pos:1370>-32.5; -48.988; 5063953;</pos:1370>
```

Timestamps der sichtbaren Objekte werden alle 32ms in den Stream geschrieben.
Es gibt zu einem Timestamp in der Regel mehrere Objekteinträge.

Es gibt seitens der Torque Engine keine Möglichkeit auf die Systemzeit zuzugreifen.
Als einzige Referenz ist es möglich den sogenannten „Click-Count“ auszulesen.
Grob formuliert handelt es sich hierbei um die Zeit welche vergangen ist seit dem die CPU aktiviert wurde. (quasi seit dem der Computer eingeschalten wurde)
Um eine wissenschaftlich verwertbare Zuordnung zu erhalten ist es erforderlich das andere Aufzeichnungsprogramme ebenso den Click-Count als Zuordnung liefern.

Tracker (Behavior)

Bei den ersten Versuchen wurde noch für jedes Objekt (Enemy, Mine, Player, ...) ein individueller Tracker erstellt.
Nun wurde die Programmlogik soweit angepasst das der Tracker bei jedem Objekt in Torque anhängt werden kann und somit –ohne adaptierung- die Aufzeichnung startet.
Als Grenzen der Aufzeichnung lassen sich inGame Koordinaten definieren.
Die Bezeichnung dient nur der besseren Lesbarkeit im Log-File.



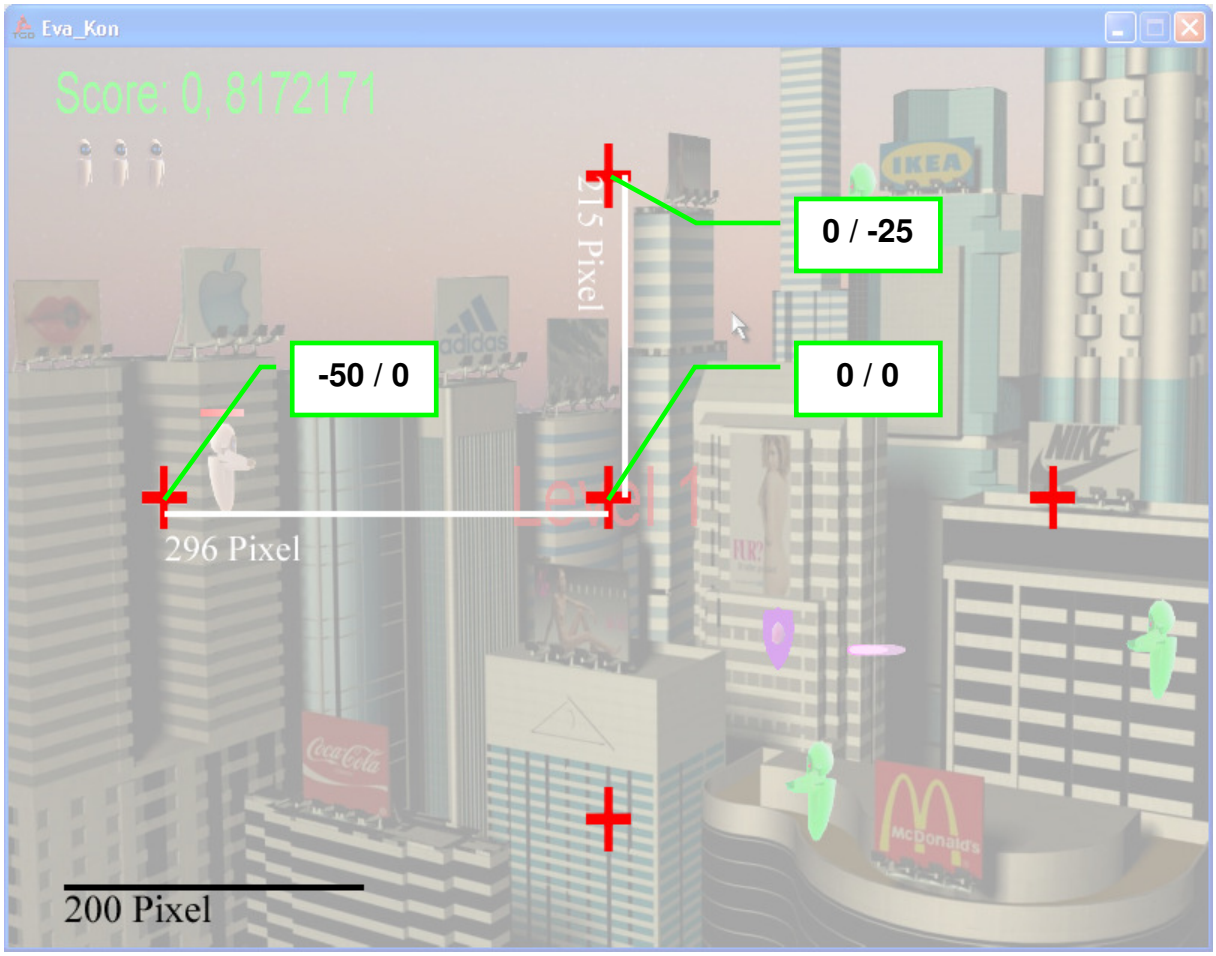
Torque Behavior, Abb. 1

Weiters ist es gelungen die Programmlogik soweit zu Generalisieren das für den Benutzer praktisch kein Eingabeaufwand erforderlich ist.
Die einzigen optionalen Einträge sind:

- desription: dient nur der besseren Lesbarkeit des Log-Files. Kann frei gewählt werden.
- loglimitX: wenn ein Objekt diese inGame-Koordinate überschreitet wird seine Trajektorie nicht mehr mit aufgezeichnet. (Objekt überschreitet +/- loglimitX)
- loglimitY: wenn ein Objekt diese inGame-Koordinate überschreitet wird seine Trajektorie nicht mehr mit aufgezeichnet. (Objekt überschreitet +/- loglimitY)

Koordinaten

inGame vs Bildschirmkoordinaten



Koordinatensystem 1

Erkenntnisse:

- der Koordinatenenursprung der inGame-Koordinaten wird sowohl bei den Objekten als auch beim Canvas zentriert in der Mitte angenommen.
- die Darstellung wird im Spiel verzerrt, so entspricht der Abstand auf ein um in y-Richtung um `25` Einheiten verschobenes Objekt zum Ursprung in Wahrheit `215` Pixel und der Abstand auf ein in x-Richtung um `50` Einheiten verschobenes Objekt `296` Pixel. (somit enspricht `1` Pixel in x-Richtung einem anderen reellen Abstand als `1` Pixel in y-Richtung da $2 * 215 Pixel \neq 296 Pixel$) [siehe Umrechnungsformel]
- Alle Objekte welche folgende Formel erfüllen befinden sich außerhalb des sichtbaren Bereiches (und sind somit nonVisible [nv]): $(abs|x| > 67{,}5) \vee (abs|y| > 35)$

Umrechnungsformel inGame zu realen Koordinaten

Dadurch ergibt sich folgende Umrechnungen von $abs|Abs\tan d|$ inGame zu $abs|Abs\tan d|$ Real:

x-Richtung: `1` Einheit inGame = `5,92` Pixel real
y-Richtung: `1` Einheit inGame = `8,6` Pixel real

Hinweis: diese Umrechnungsformel wird nicht verwendet da die Poisitonsangaben des Eye-Trackers zwischen 0 und 1 liegen.
(Sprich linkes oberes Eck des Bildschirms ist 0 / 0, rechtes unteres 1 / 1)

deswegen wird folgende Formel verwendet:

Umrechnungsformel inGame zu Eye-Tracker Koordinaten

Die Umrechnung der inGame Koordinaten auf die 0/1 Eye-Tracker Koordinaten erfolgt durch folgende Formel:

x-Koordinaten: $\frac{x+67{,}5}{135}$ [0,1]

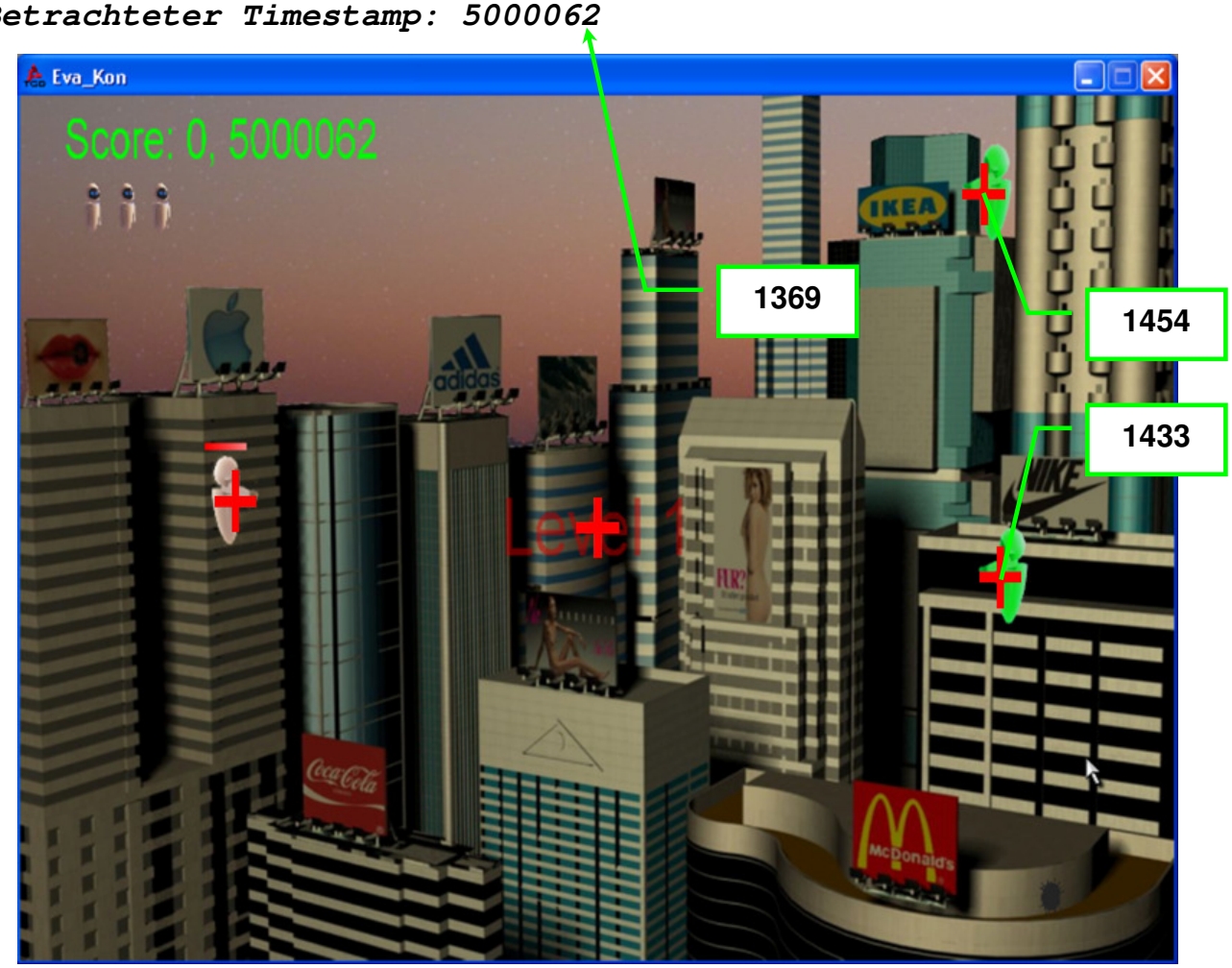
y-Koordinaten: $\frac{y+35}{70}$ [0,1]

Info:
- Die Abmessungen des Spiels sind 135 x 70 Pixel (inGame-Pixel)
- der Koordinaten-Ursprung befindet sich in der Mitte (inGame)
- die linke obere Ecke des Fensters liegt somit bei -67,5 / -35 (inGame)

Auswertungen

Auswertung 1

Betrachteter Timestamp: 5000062



Auswertung 1- 1

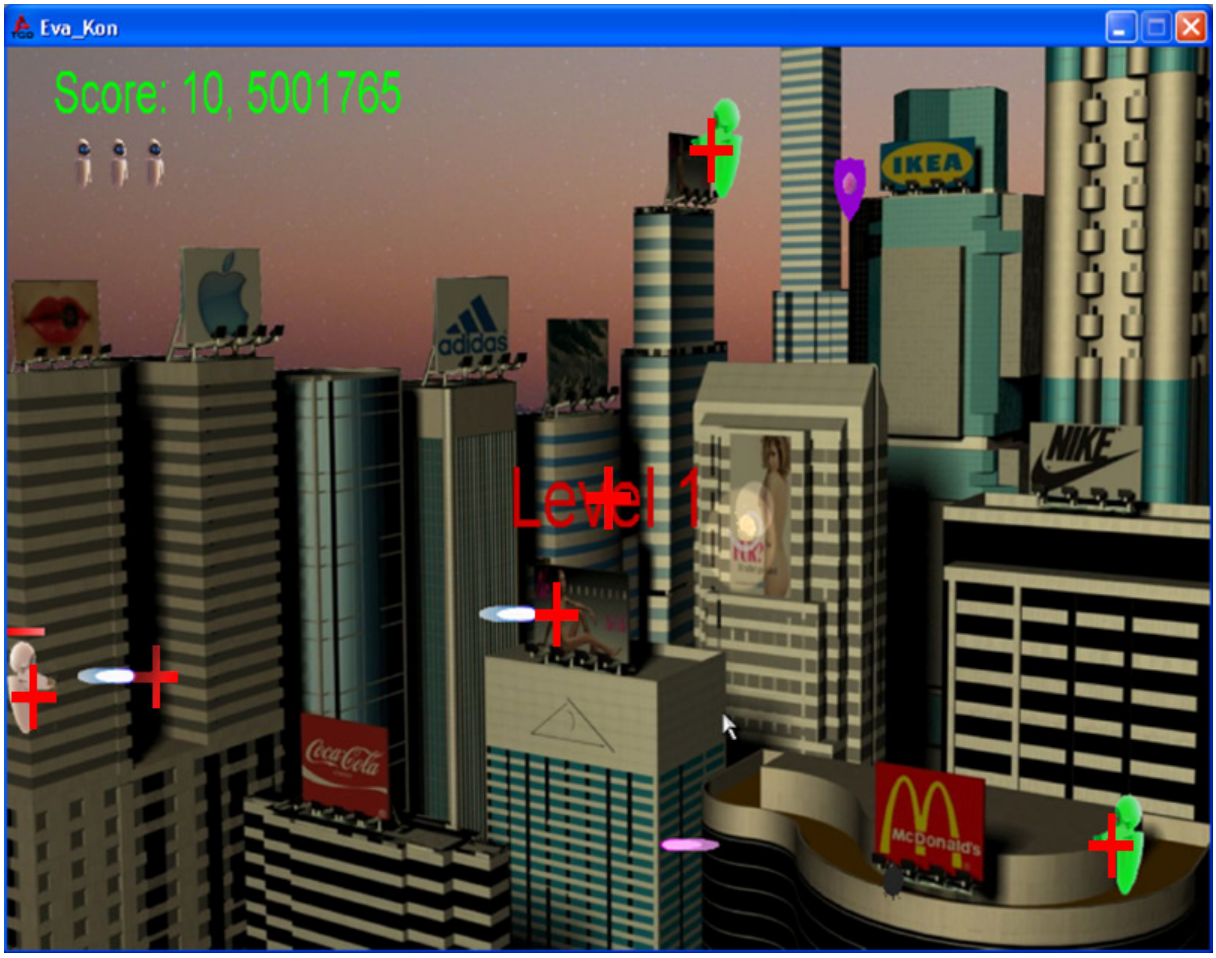
Hinweise zu Objekten:

```
<newObject:1355>1355</newObject:1355>
<name:1355>PLAYR</name:1355>
<sizeX:1355>5.638</sizeX:1355>
<sizeY:1355>7.603</sizeY:1355>

<newObject:1433>1433</newObject:1433>
<name:1433>ENEMY</name:1433>
<sizeX:1433>6</sizeX:1433>
<sizeY:1433>8.326</sizeY:1433>

<newObject:1454>1454</newObject:1454>
<name:1454>ENEMY</name:1454>
<sizeX:1454>6</sizeX:1454>
<sizeY:1454>8.326</sizeY:1454>
```

Datensätze zum Timestamp 5000062:

```
<pos:1355>-42.431; -2.109; 5000062;</pos:1355>      //Player
<pos:1369>-32.5; -48.988; 5000062;</pos:1369>       //Player Laser nv
<pos:1433>47.1652; 4; 5000062;</pos:1433>           //Enemy 1433
<pos:1454>45.2579; -27; 5000062;</pos:1454>         //Enemy 1454
```

Betrachteter Timestamp: 5001765



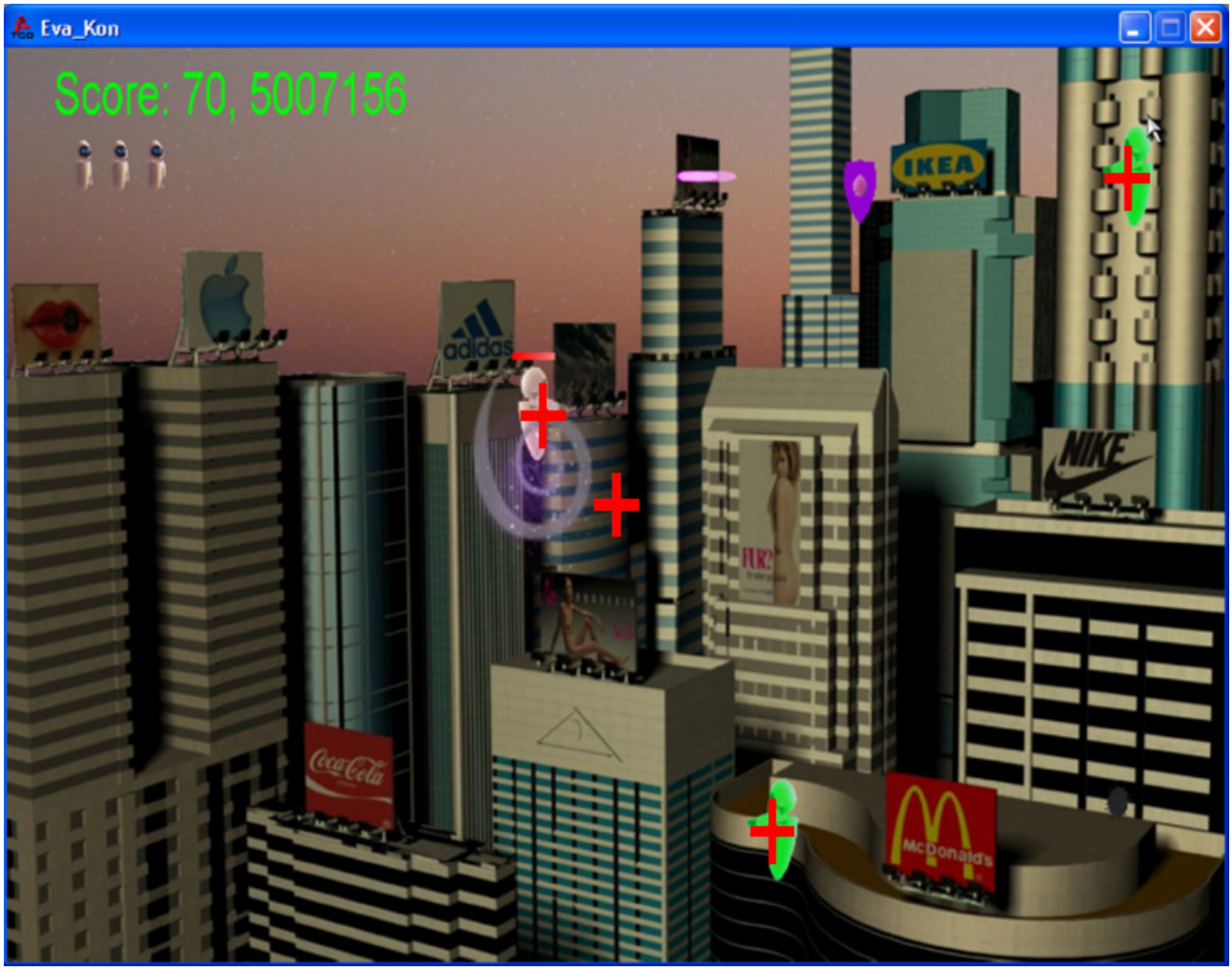
Auswertung 1- 2

Hinweis: Für die Betrachtung / Ausgabe wurden nur die Datensätze zu „Player“, „Player-Laser“ und „Enemy“ aktiviert. (für mehr Übersichtlichkeit bei der ersten Auswertung)
Deshalb scheinen die „Enemy-Laser“ / das „Health-Object“ bei dieser händischen Auswertung noch nicht auf obwohl sie in Abb. 2, Auswertung 1-2 zu sehen sind.

Datensätze zum Timestamp 5001765:

```
<pos:1355>-64.68; 15.491; 5001765;</pos:1355>      //Player
<pos:1369>-32.5; -48.988; 5001765;</pos:1369>      //Player Laser nv
<pos:1454>11.5771; -27; 5001765;</pos:1454>        //Enemy 1454
<pos:1475>56.5609; 27; 5001765;</pos:1475>         //Enemy 1475
<pos:1487>-5.791; 9.091; 5001765;</pos:1487>       //Player Laser
<pos:1496>-50.911; 13.891; 5001765;</pos:1496>     //Player Laser
```

Betrachteter Timestamp: 5007156



Auswertung 1- 3

Datensätze zum Timestamp 5007156:

<pos:1355>-8.11624; -6.84202; 5007156;</pos:1355>	//Player
<pos:1369>-32.5; -48.988; 5007156;</pos:1369>	//nv
<pos:1454>-95.8204; -27; 5007171;</pos:1454>	//nv
<pos:1487>1075.81; 9.091; 5007171;</pos:1487>	//nv
<pos:1496>1030.69; 13.891; 5007171;</pos:1496>	//nv
<pos:1514>904.601; 14.691; 5007171;</pos:1514>	//nv
<pos:1517>862.361; 20.291; 5007171;</pos:1517>	//nv
<pos:1556>767.321; 28.291; 5007171;</pos:1556>	//nv
<pos:1559>740.921; 24.291; 5007171;</pos:1559>	//nv
<pos:1562>698.68; 17.891; 5007171;</pos:1562>	//nv
<pos:1586>626.68; 7.49105; 5007171;</pos:1586>	//nv
<pos:1598>17.3209; 25; 5007171;</pos:1598>	//Enemy 1598
<pos:1610>416.44; -14.109; 5007171;</pos:1610>	//nv
<pos:1625>371.64; -10.909; 5007171;</pos:1625>	//nv
<pos:1661>253.987; -7.25598; 5007171;</pos:1661>	//nv
<pos:1670>215.765; -7.48798; 5007171;</pos:1670>	//nv
<pos:1679>56.7503; -25; 5007171;</pos:1679>	//Enemy 1679

Beobachtungen / Auswertungsergebnis

Die roten Kreuze wurden nachträglich anhand der aufgezeichneten Log-File Koordinaten eingetragen.
Es deutet darauf hin dass die Genauigkeit der Koordinaten-Einträge absolut ausreichend ist für eine genaue Auswertung.

Nach 7 Sekunden Spielzeit ($(5007165ms - 5000062ms) * 10^{-3} \approx 7\,\text{sec}$) sind bereits mindestens 14 Leereinträge vorhanden. (nv)
Hier soll noch eine geeignete Programmlogik gefunden werden welche die Datensätze bereits in Echtzeit aus dem Stream herausfiltert.

2Do / Infos:

- die Laserstrahlen des Players werden scheinbar nicht „mittig“ aufgezeichnet sondern an deren äußerst rechten x-Position. (y ist korrekt zentriert) -siehe Abb. 2, 1-2
- ob sich diese Verschiebung auch bei den Enemy-Lasern ergibt muss noch eruiert werden. (da diese Daten bei dieser Aufzeichnung deaktiviert wurden)
- Objekte welche sich nicht sichtbar am Bildschirm befinden werden dennoch mit aufgezeichnet. Hier soll überprüft werden ob sich diese Datensätze noch vor dem Eintrag in das Log-File ausblenden lassen. (Probleme hinsichtlich Performance müssen berücksichtigt werden da hierzu noch mehr zusätzliche Logik erforderlich ist!)

Auswertung 2

Hinweis: nun wurden alle Objekte mit aufgezeichnet. Objekte welche sich nicht mehr auf dem Canvas befinden werden nicht mehr mit aufgezeichnet. (deleteObject Eintrag im Log-File)

Betrachteter Timestamp: 5224515

Auswertung 2- 1

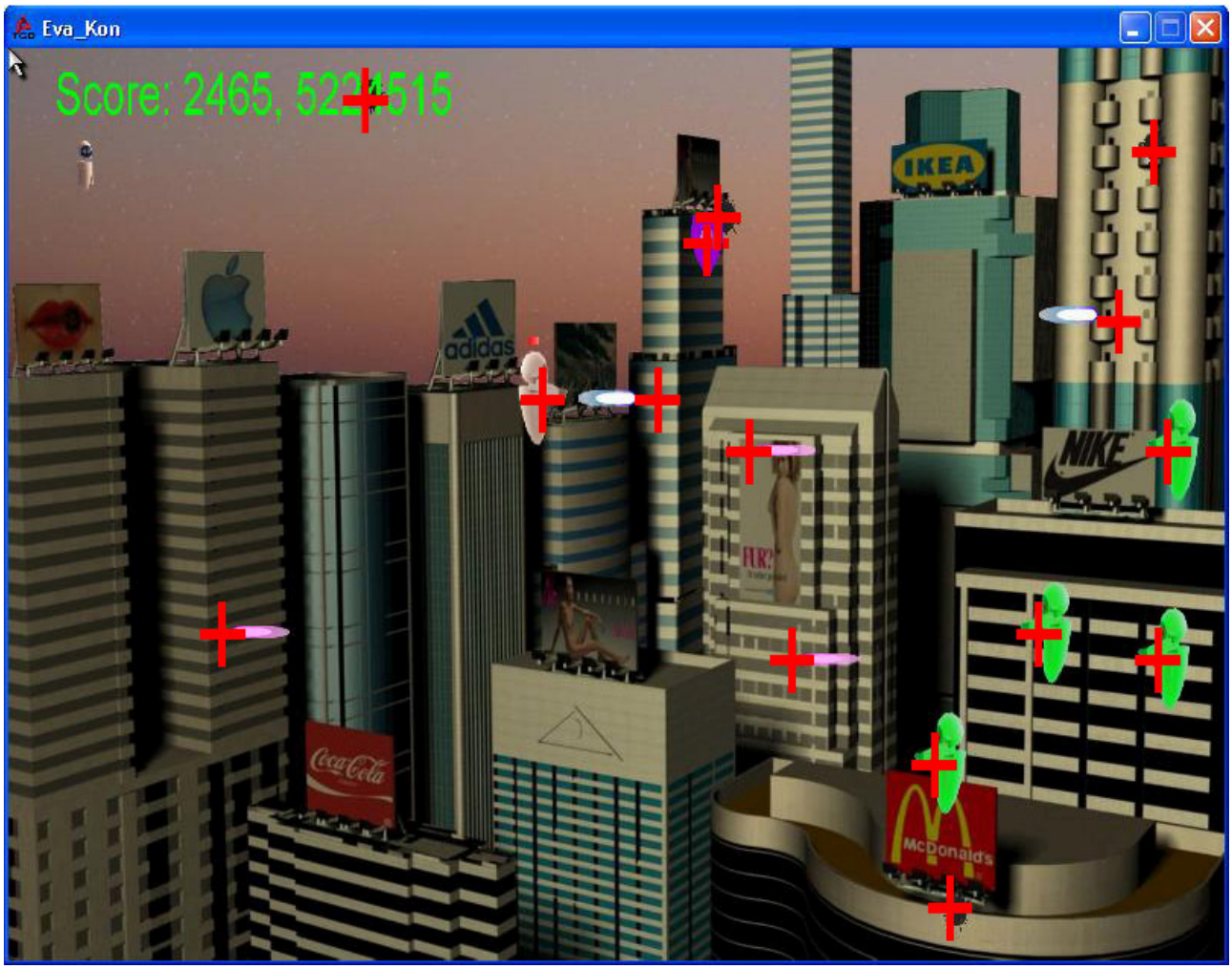
Datensätze zum Timestamp 5224515:

```
<pos:1356>-8.20949; -8.00249; 5224515;</pos:1356>          //Player
<pos:1370>-32.5; -48.988; 5224515;</pos:1370>              //Player-Laser
<pos:1419>39.387; -43.691; 5224515;</pos:1419>             //Enemy-Laser
<pos:9995>-27.8538; -31; 5224515;</pos:9995>               //Mine
<pos:10350>11.2925; -22; 5224515;</pos:10350>              //Mine
<pos:10508>10; -20; 5224515;</pos:10508>                   //Health-Item
<pos:10536>37.0441; 31; 5224515;</pos:10536>               //Mine
<pos:10672>35.2933; 20; 5224515;</pos:10672>               //Enemy
<pos:10706>46.9007; 10; 5224515;</pos:10706>               //Enemy
<pos:10715>-43.758; 9.94817; 5224515;</pos:10715>          //Enemy-Laser
<pos:10719>59.6483; -27; 5224515;</pos:10719>              //Mine
<pos:10753>61.2143; -4; 5224515;</pos:10753>               //Enemy
```

```
<pos:10762>14.7; -4; 5224515;</pos:10762>          //Enemy-Laser
<pos:10766>55.791; -14.402; 5224515;</pos:10766>   //Player-Laser
<pos:10769>60.091; 12; 5224515;</pos:10769>        //Enemy
<pos:10778>19.5; 12; 5224515;</pos:10778>          //Enemy-Laser
<pos:10790>4.591; -8.002; 5224515;</pos:10790>     //Player-Laser
```

Anhand dieser Koordinaten wurden jetzt nachträglich Kreuz-Symbole in das Spiel eingetragen:

Die Überdeckungen zeigten eindeutig dass die Aufzeichnung genau ist.
(Programm ca. 3 Minuten lang ausgeführt, ein 3 MB großes Log-File mit über 65.000 Einträgen wurde erzeugt.
Die hier betrachteten Einträge befinden sich an Position 64.857 –sind also Einträge welche erst relativ spät hinzugekommen sind.)



Auswertung 2- 2

2Do / Infos:

- die Laserstrahlen von Player und Enemys werden nicht zentriert angegeben.
(Konstanter Verschiebungsfaktor in x, y korrekt zentriert)

Zukünftige Aufgaben

- die Laserstrahlen von Player und Enemys werden nicht zentriert angegeben. (Konstanter Verschiebungsfaktor in x, y korrekt zentriert)
- Grafische Auswertung der Ergebnisse (Momentan werden die Ergebnisse nur auf der Konsole ausgegeben) z.B. Histogramm
- Implementierung unserer Funktionalität in die Applikation „Iyzer“
- Experimente mit unterschiedlichem Aufbau durchführen
- Bestimmen ob die Genauigkeit des Eye-Trackers ausreicht

Abbildungsverzeichnis