Entwicklung und Konfiguration von visuellen Anzeigen für die Objektsteuerung im Projekt ArbeitsTeilung Entwickler-Operateur (ATEO)

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von:	Aydan Seid
geboren am:	15.01.1983
in:	Dulovo (Bulgarien)
Gutachter:	Prof. Dr. Klaus Bothe
	Prof. Dr. Hartmut Wandke
eingereicht am:	

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

AKRONYME

ATEO Arbeitsteilung Entwickler-Operateur

prometei Prospektive Gestaltung von Mensch-Technik-Interaktion

SAM Socially Augmented Microworld

AAF ATEO Automation Framework

AAFGT ATEO Automation Framework Graphic Tool

AMD ATEO Master Display

OA Operateursarbeitsplatz

MWB Mikroweltbewohner

XML Extensible Markup Language

GUI Graphical User Interface

1 EINLEITUNG

Dieses Kapitel dient zum Einen der Vorstellung des Projekts, in dessen Rahmen diese Diplomarbeit angesiedelt ist und zum Anderen der Formulierung der Ziele der vorliegenden Arbeit. Zum Schluss wird noch auf die Gliederung der nachfolgenden Kapitel eingegangen.

1.1 DAS ATEO-PROJEKT

Das Projekt Arbeitsteilung Entwickler-Operateur (ATEO) ist ein interdisziplinäres Projekt vom Graduiertenkolleg Prospektive Gestaltung von Mensch-Technik-Interaktion (prometei) der Technischen Universität Berlin und vom Lehrstuhl Ingenieurpsychologie und Kognitive Ergonomie des Instituts für Psychologie an der Humboldt-Universität zu Berlin. Ziel des ATEO-Projekts ist die Untersuchung der Funktionsteilung zwischen Mensch und Maschine, d.h. welche Aufgaben hinsichtlich eines dynamischen, komplexen, technischen Systems[1] können besser von einem menschlichen Operateur bzw. von einer von Entwicklern konzipierten und implementierten Automatik erledigt werden. Die Führung und die Überwachung von Prozessen steht dabei im Mittelpunkt der Untersuchung. Die gewonnenen Erkenntnisse können künftig bei der Planung und der Gestaltung von derartigen Systemen miteinbezogen werden.

Für die Durchführung der Untersuchung wurden im Rahmen des Projekts bereits die Versuchsumgebung Socially Augmented Microworld (SAM) sowie der Operateursarbeitsplatz ATEO Master Display (AMD), welches zur Überwachung und zum Eingriff durch den Operateur dient, implementiert. Für die Entwicklung von Automatiken und deren Einbindung in SAM, um dadurch die Untersuchung der Arbeit von Entwicklern im Vergleich zum Operateur zu ermöglichen, wurde das ATEO Automation Framework (AAF) konzipiert und implementiert.

Im Folgenden erfolgt eine kurze Vorstellung der drei oben erwähnten Software-Komponenten von ATEO. In Abbildung 1 ist deren Zusammenwirken visualisiert.

1.1.1 *Socially Augmented Microworld (SAM)*

SAM stellt die Versuchsumgebung von ATEO dar und ist durch Bothe u. a. (2010) spezifiziert. Es handelt sich hierbei um eine Trackingaufgabe, bei der zwei Personen, die sogenannten Mikroweltbewohner

1 Z.B. Flugzeugsteuerung, Produktionsprozesse u.ä.

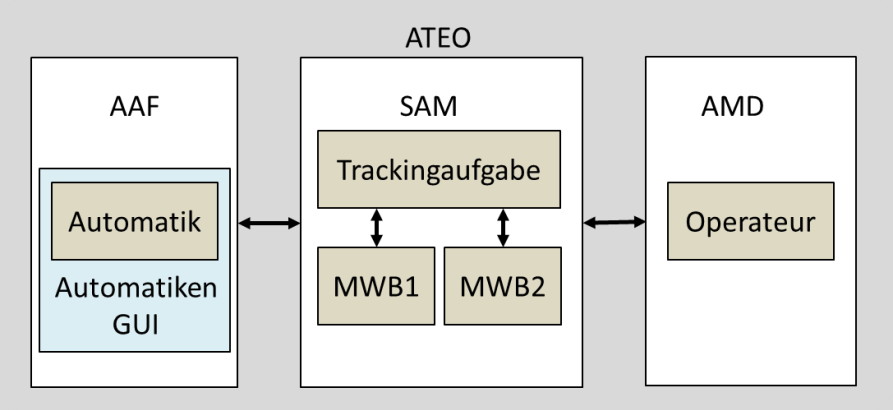

Abbildung 1: ATEO Software-Komponenten

(MWB), gemeinsam durch ihre Joystickeingaben ein rundes Objekt entlang einer vorgegebenen Strecke bis zum Ziel steuern müssen (Abbildung 2). Dabei wird die Steuerung beider MWB im Normalfall zu gleichen Anteilen in die Objektsteuerung einbezogen. Diese können jedoch durch den Operateur oder durch die eingesetzte Automatik, welche die MWB bei der Durchführung ihrer Aufgabe unterstützen sollen, individuell verändert werden.

Die primäre Aufgabe der beiden MWB besteht darin, das Objekt so schnell wie möglich und so genau wie möglich vom Start bis zum Ziel zu navigieren. Zusätzlich bekommt jeder MWB eine individuelle Aufgabe gestellt, dass er seine Priorität auf die Schnelligkeit bzw. auf die Genauigkeit des Fahrens setzen soll. Die Zweitaufgabe und das bis zum gewissen Maße nichtdeterministische[2] Verhalten der MWB sowie das Vorhandensein von Gabelungen und Hindernissen auf der Strecke (Abbildung 2) führen zur Entstehung von Konfliktsituationen, die beiden MWB dürfen sich während der Fahrt nicht absprechen, was der Erhöhung der Dynamik und der Komplexität vom System dient. Somit kann SAM den komplexen Prozess simulieren, der für die Untersuchung der in diesem Abschnitt dargelegten Fragestellung des Projekts ATEO benötigt wird.

1.1.2 *ATEO Automation Framework (AAF)*

Das AAF, entwickelt von Hasselmann (2012, in Bearb.) in seiner Diplomarbeit, ist die Software-Komponente von ATEO, die den Entwicklern das Erstellen von Automatiken und deren Integration in SAM ermöglicht. Das Framework sieht vor, dass jede Automatik idealerweise eine einzige, klar definierte Aufgabe erfüllt, wie z.B. Bremsen oder Lenken und wird deshalb auch als Funktion bezeichnet. Durch diese Eigenschaft können elementare Automatiken oder Funktionen

2 Man kann annehmen, dass sich die Personen während der Fahrt an ihre Anweisungen halten werden, aber sie müssen es nicht immer tun.

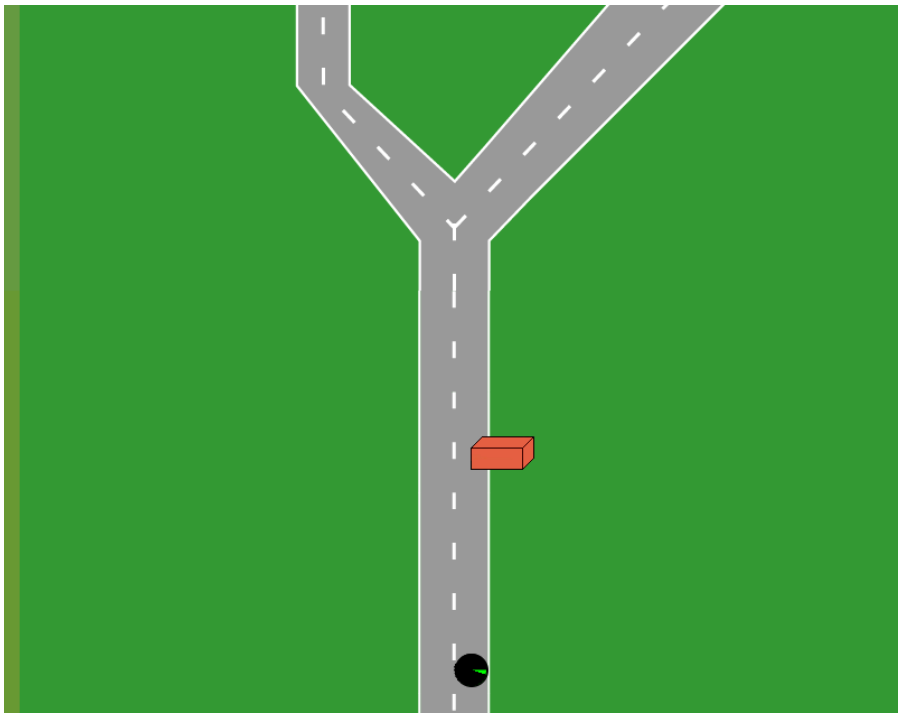

Abbildung 2: SAM: Trackingobjekt auf einer Strecke mit Gabelung und Hindernis

als Bausteine für die Erstellung von komplexeren Automatiken benutzt werden. Für diesen Zweck wird jede Automatik, elementar oder komplex, in AAF als ein Graph modelliert. Jeder Graph hat einen Wurzel- und ein Senkenknoten. Als innere Knoten können sowohl elementare als auch bereits zusammengesetzte, komplexe Automatiken eingesetzt werden. Dabei hat jeder innere Knoten genau einen Vorgänger und genau einen Nachfolger. Die beiden Endknoten haben keinen Vorgänger und beliebig viele Nachfolger respektive keinen Nachfolger und beliebig viele Vorgänger und stellen die Schnittstelle zwischen AAF und SAM dar. Der aktuelle Zustand von SAM wird über den Wurzelknoten in den Graph der aktiven Automatik eingespeist. Jeder innere Knoten kann von seinem Vorgänger übergebenen Zustand verändern und damit unter Umständen seine Änderungen überschreiben. Der neue Zustand wird nach der Abarbeitung des Graphen durch die Senke wieder in SAM geschrieben. Auf diese Weise finden die Eingriffe einer Automatik in den Prozess von SAM statt.

Um Automatiken in AAF zusammenbauen zu können, muss man sich sowohl mit dem Framework als auch mit der Sprache Smalltalk, die Sprache in der alle Software-Komponenten in ATEO implementiert sind, auskennen. Aus diesem Grund können Personen ohne entsprechende Kenntnisse, z.B. Versuchsleiter, keine Automatiken bauen und benutzen. Hierfür hilft das ATEO Automation Framework Graphic Tool (AAFGT).

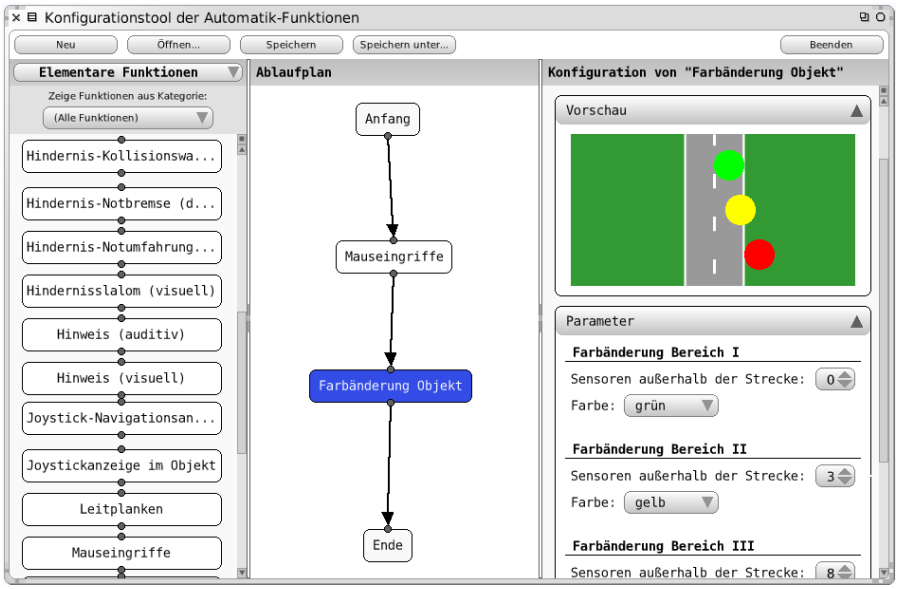

Abbildung 3: AAFGT: Beispielgraph einer Automatik

ATEO Automation Framework Graphic Tool (AAFGT)

Das grafische Tool AAFGT (Abbildung 3) entstand im Rahmen der Diplomarbeit von Fuhrmann (2010) und wurde in der Studienarbeit von Kosjar (2011) auf seine Gebrauchstauglichkeit überprüft und überarbeitet. Es erlaubt dem Anwender das einfache und intuitive Erstellen von Automatiken. Für diesen Zweck werden ihm alle im System vorhandenen Automatikfunktionen als Bausteine bereitgestellt, die er zu einer Automatik nach seinem Wunsch zusammenbauen kann. Die einzelnen Automatikfunktionen können dabei durch Veränderung ihrer Parameter konfiguriert werden. Eine Aktivierung der Automatik oder ihrer Funktionen für bestimmte Streckenabschnitte oder in Abhängigkeit des Auftretens eines konfigurierbaren Ereignisses ist auch möglich. Die fertige Automatik wird abgespeichert und kann somit als Bauteil in anderen Automatiken verwendet oder zu einem späteren Zeitpunkt geladen und verändert werden.

1.1.3 *ATEO Master Display (AMD)*

Für die Überwachung und Führung des SAM-Versuchslaufs durch den Operateur wurde das AMD, früher Operateursarbeitsplatz (OA) genannt, implementiert. Da diese ATEO-Komponente für die vorliegende Arbeit keine Relevanz hat, verzichtet der Autor auf ihre Vorstellung. Für eine detaillierte Beschreibung wird auf die Diplomarbeit von Leonhard (2012) hingewiesen.

1.2 MOTIVATION

Wie bereits im Abschnitt 1.1 beschrieben, beschäftigt sich das Projekt ATEO mit der Identifizierung von Aufgabenklassen im Kontext von Prozessüberwachung und -führung innerhalb eines komplexen, technischen Systems, die prinzipiell besser durch einen menschlichen Operateur bzw. eine von Entwicklern konzipierten und implementierten Automatik gelöst werden können. Für die Bewertung der Leistung des Operateurs wurden bereits Untersuchungen durchgeführt. Um dies auch mit Automatiken tun zu können, damit der Beitrag der Entwickler in die Untersuchung eingebracht werden kann, müssen diese noch implementiert werden. Die vorliegende Arbeit leistet neben der Diplomarbeit von Kosjar (2012), der Studienarbeit von Wickert (2012a) sowie seiner Diplomarbeit (Wickert (2012b, in Vorb.)) einen Beitrag dazu.

1.3 ZIEL DER ARBEIT

Im Vorfeld dieser Arbeit hat Kain (2012, in Vorb.) 30 Teams aus verschiedenen Industriebranchen wie Fahrzeugbau, Luftfahrt und Softwareentwicklung sowie Teams aus Universitäten die Aufgabe gestellt, Automatiken für die Prozessüberwachung und -führung in SAM zu konzipieren. Die von den Teams entwickelten Automatikkonzepte und deren Funktionen wurden von selbigen auf die sogenannten *Konzeptbögen* aufgeschrieben. Diese Konzeptbögen wurden anschließend von Kain in tabellarischer Form digitalisiert.

Das Ziel dieser Arbeit ist die Analyse der erwähnten Konzeptbögen und die Implementierung all derjenigen Automatikfunktionen, die eine visuelle Darstellung der aktuellen und optimalen Objektposition, -richtung und -geschwindigkeit oder das Anzeigen der aktuellen und optimalen Joystickposition der MWB beinhalten. Es sind Automatikfunktionen, die die MWB unterstützen, ohne direkt in die Steuerung einzugreifen und werden daher als weiche Eingriffe[3] bezeichnet. Die Integration der Funktionen ins im Abschnitt 1.1.2 vorgestellte AAFGT, d.h. das Bereitstellen eines Konfigurationsdialogs für die jeweilige Funktion für den Endanwender, gehört auch zum Ziel der Arbeit. Alle restlichen in den Konzepten vorkommenden Funktionen mit weichen Eingriffen, zu denen auch auditive Hinweise gehören, werden in der Diplomarbeit von Kosjar (2012) behandelt.

Diese Arbeit basiert auf den Automatikkonzepten aus der Arbeit von Kain (2012, in Vorb.), sowie auf dem Framework von Hasselmann (2012, in Bearb.), zur Integration der Entwicklerautomatiken in SAM und auf dem darauf aufbauenden AAFGT von Fuhrmann (2010).

3 Eingriffe in die Objekt- oder MWB-Steuerung gelten entsprechend als hart.

1.4 GLIEDERUNG DER ARBEIT

Kapitel 2 erläutert die technischen und ATEO-Software spezifischen Grundlagen, die zum Verständnis der in dieser Arbeit angestrebten Entwicklung vorausgesetzt werden. Die Analyse der Bögen mit den Automatikkonzepten, die im Vorfeld dieser Arbeit erstellt wurden, und ihre Ergebnisse werden in Kapitel 3 beschrieben. Die Erläuterung der Umsetzung der durch die Analyse ausgewählten Automatikfunktionen in Smalltalk findet in Kapitel 4 statt. Die Tests zur Validierung der Implementierung werden in Kapitel 5 behandelt. In Kapitel 6 wird das Verfahren für die Bewertung der Automatikanzeigen vorgestellt und im abschließenden Kapitel 7 werden die Ergebnisse der vorliegenden Arbeit zusammengefasst.

Anhang A enthält die Liste mit den Namen der Klassen, die im Rahmen dieser Arbeit implementiert wurden. Die Ergebnisse der Bewertung der Automatikanzeigen sind in Anhang B, während die Konfigurationsmöglichkeiten für jede implementierte Automatik in Anhang C zu finden sind. Der Inhalt der beigelegten DVD ist in Anhang D beschrieben.

2 VORAUSSETZUNGEN

Dieses Kapitel erklärt wichtige technische und projektspezifische Voraussetzungen, die zum Verständnis der nachfolgenden Kapitel notwendig sind.

Es beginnt mit einer Einführung in die Programmiersprache Smalltalk, die Sprache, in der die gesamte Software von ATEO implementiert wird. Hier wird auch die im Projekt verwendete Entwicklungsumgebung Squeak sowie die Werkzeuge von Smalltalk für Realisierung von grafischen Bildschirmausgaben kurz vorgestellt. Danach erfolgt die Anleitung für die Implementierung neuer Automatikfunktionen in AAF. Anschließend wird auf die Besonderheiten und die Probleme von SAM eingegangen, die bei der Implementierung von Automatiken beachtet werden müssen.

2.1 SMALLTALK

Die Entwicklung der Programmiersprache Smalltalk[1] begann in den frühen 70er Jahren am Palo Alto Research Center (PARC) von Xerox. Nach einer etwa 10-jährigen Weiterentwicklung wurde die Sprache unter dem Namen Smalltalk-80 in der Version 1 außerhalb von Xerox veröffentlicht (Goldberg u. a., 1983). Maßgeblich an der Entstehung und der Entwicklung der Sprache haben vor allem Alan Kay, Adele Goldberg und Dan Ingalls mitgewirkt. Praktisch alle der heute vorhandenen Implementierungen von Smalltalk, kommerziell oder frei verfügbar, basieren auf Smalltalk-80.

Smalltalk hat einen sehr kleinen Sprachumfang und ist daher sehr einfach und schnell zu erlernen. Es ist eine rein objektorientierte Programmiersprache, d.h. alles in Smalltalk ist ein Objekt selbst Zahlen und Zeichen. Dem Programmierer stehen nur folgende Möglichkeiten bei der Arbeit mit Objekten zu Verfügung:

- Definition von Variablen,
- Zuweisung von Objekten an Variablen,
- Lieferung eines Objektes als Rückgabewert einer Methode,
- Senden einer Nachricht an ein Objekt.

Die Variablen werden dynamisch typisiert. Bei ihrer Definition wird kein Datentyp angegeben (wie z.B. es bei Java oder C++ der Fall ist),

1 http://www.smalltalk.org

sondern er wird erst zur Laufzeit vom ihr zugewiesenen Objekt abgeleitet. Die Interaktionen zwischen den Objekten finden durch das Senden von Nachrichten statt. Auf diese Weise werden auch die von anderen Sprachen bekannten Kontrollstrukturen wie bedingte Verzweigungen oder Bedingungs- und Zählschleifen realisiert. Hierfür werden die entsprechenden Nachrichten an ein boolesches Objekt (bedingte Verzweigung und Bedingungsschleife) oder an eine Zahlenmenge (Zählschleifen) gesendet.

Die Eingabe von neuem Programmcode geschieht in der Sprache selbst, weil jede Implementierung von Smalltalk neben der Laufzeitumgebung für die Sprache auch eine integrierte grafische Entwicklungsumgebung mit sich bringt. In dieser können die Entwickler ihre Programme editieren, debuggen und ausführen. Die Kompilierung des Codes wird beim Speichern des Programms automatisch durch die Entwicklungsumgebung durchgeführt.

2.1.1 *Squeak*

Squeak[2] ist die moderne, quelloffene und auf allen gängigen Betriebssystemen[3] laufende Implementierung von Smalltalk-80 samt Entwicklungsumgebung, die für die Implementierung der ATEO-Software verwendet wird. Entwickelt von einer Arbeitsgruppe bei Apple Computer, darunter auch einige der ursprünglichen Entwickler von Smalltalk-80 wie Alan Kay und Dan Ingalls, wurde 1996 die erste Version von Squeak als erstes freies Smalltalk-System veröffentlicht. Mittlerweile wird die Entwicklung von Squeak von einer offenen Community unterstützt und vorangetrieben.

Squeak kommt mit einer großen Auswahl an Klassenbibliotheken wie z.B. für die Entwicklung von Benutzeroberflächen, für das Abspielen von Musik und Videos u.v.m., die dem Entwickler viel Programmierarbeit abnehmen. Darüber hinaus kann durch die Installation von über 780 Erweiterungen[4] die Funktionalität von Squeak an die eigenen Wünsche angepasst werden.

Obwohl die Entwicklung von Squeak sich aktuell in der Version 4.3 befindet, wird im ATEO-Projekt weiterhin die bereits fünf Jahre alte Version 3.9.2 (Abbildung 4) eingesetzt. Der Grund dafür ist, dass die neueren Squeak-Versionen abwärtsinkompatibel sind und deshalb eine Portierung der Projekt-Software mit einem nicht unerheblichem Aufwand verbunden oder gar nicht möglich ist.

Für den schnellen Einstieg in die Programmierung mit Squeak, aber auch für fortgeschrittene Anwender interessant, ist das offene Buch "Squeak by Example" (Black u. a., 2007) zu empfehlen. Das Web-

2 http://www.squeak.org
3 Windows, Mac OS, Unix und verschiedene Linux-Distributionen
4 http://map.squeak.org/

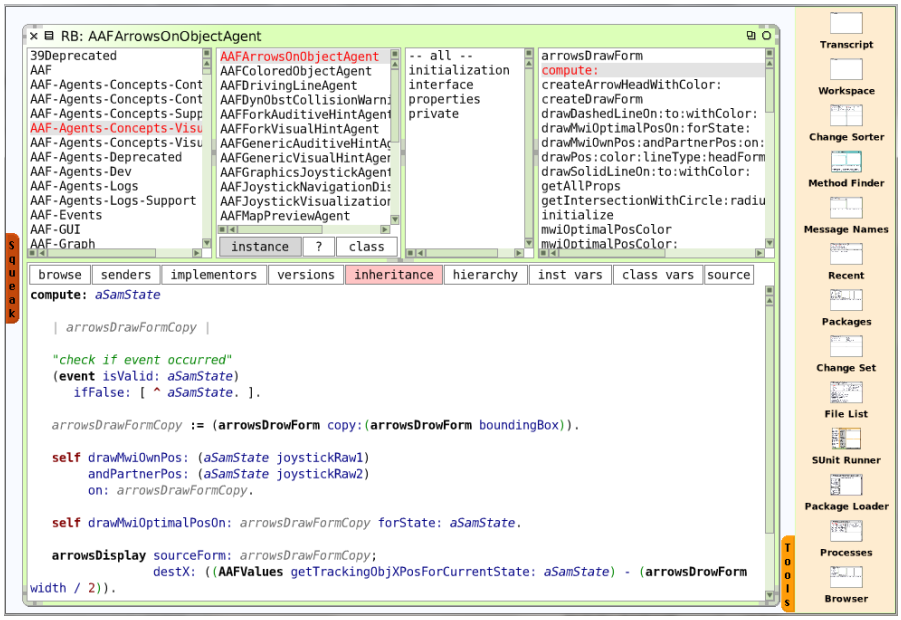

Abbildung 4: Squeak: Browser- und Editor-Fenster und Werkzeugleiste

Kompendium *LanguageNotes...*[5] sollte auch jeder Squeak-Entwickler kennen.

2.1.2 *Grafische Elemente*

Das Ziel der vorliegenden Arbeit ist die Implementierung von Automatiken, die durch ihre visuellen Ausgaben die MWB bei der Ausführung ihrer Trackingaufgabe unterstützen sollen. Aus diesem Grund werden in diesem Abschnitt die Elemente von Smalltalk/Squeak vorgestellt, die den Entwicklern dafür zur Verfügung stehen.

Als Erstes ist hier das Grafiksystem von Squeak namens *Morphic* (Black u. a., 2007, S.231-251) zu erwähnen. Es wurde mit dem Ziel entwickelt, dem Programmierer den Umgang mit grafischen Objekten so viel wie möglich zu erleichtern. Das geschieht allerdings auf Kosten der Performanz. Die Grafikobjekte von Morphic, genannt *Morphe*, sind träge und zu langsam für die Erzeugung der grafischen Ausgaben von Automatiken. Ein Simulationsschritt in SAM dauert im Durchschnitt 39ms (mehr dazu im Abschnitt 2.3). In diesem Zeitintervall muss neben der Strecke selbst auch die Anzeige der aktiven Automatik aktualisiert werden, was mit Morphen nicht zu erreichen ist. Aus diesem Grund wurde der grafische Teil von SAM (Abbildung 2) mit Hilfe von *Forms* und *BitBlts*, zwei Basiselemente aus der Grafikbibliothek von Smalltalk, realisiert. Diese werden auch für die Implementierung der Anzeigen von den Automatiken in dieser Arbeit verwendet.

5 http://squeak.joyful.com/LanguageNotes

Forms

Bilder werden im Smalltalk durch Instanzen der Klasse `Form` repräsentiert (Goldberg und Robson, 1983). Jede Form hat eine Länge, eine Breite und eine Bitmap, die den Inhalt des Bildes enthält. Der Bildschirminhalt des Rechners ist auch eine `Form`-Instanz. Auf sie wird über das globale Objekt `Display` zugegriffen. Dieses ist die einzige Instanz der Klasse `DisplayScreen`, die eine Unterklasse von `Form` ist.

Die einfachste Methode für die Instanziierung der Klasse `Form` ist der Aufruf der Klassenmethode `fromFileNamed:` (Listing 1).

```
1 form := Form fromFileNamed: 'resouce.images\tachometer.png'
```

Listing 1: Laden einer Bild-Datei in ein Form-Objekt

Die Methode lädt die Datei, deren Pfad als Parameter übergeben wurde, in ein `Form`-Objekt. Der Dateipfad kann dabei relativ zum Ordner, wo Squeak ausgeführt wird, oder absolut angegeben werden. Als Bildformat werden alle verbreiteten Formate wie PNG, GIF, JPEG oder BMP unterstützt.

Eine zweite Methode zur Erzeugung von `Form`-Objekten ist in Listing 2 zu sehen. Hier wird im Unterschied zu der ersten Methode das Objekt sozusagen per Hand erzeugt. Der Programmierer muss dafür die Länge, die Breite und die Farbtiefe des Bildes (hier wird die Farbtiefe des Bildschirms verwendet) angeben. Dadurch wird ein rechteckiges leeres Bild erzeugt, welches mit `fillColor: (Color blue)` mit blauer Farbe gefüllt wird. Die auf diese Weise erzeugten Bilder sind nicht besonders anspruchsvoll und werden vorwiegend als Hintergrund verwendet. Zwei weitere wichtige Operationen auf Objekte von der Klasse `Form` sind das Drehen und das Skalieren (Zeile 3-4) der enthaltenen Bilder.

```
1 form := Form extent: 200@100 depth: (Display depth).
2 form fillColor: (Color blue).
3 form rotateBy: 90.
4 form scaledToSize: 100@50.
```

Listing 2: Erzeugung eines Form-Objekts ohne Bilddatei

Mit den bisher vorgestellten Methoden können zwar Bilder geladen, erzeugt oder elementar manipuliert werden, aber diese führen nicht zu einer Darstellung des Bildes auf dem Bildschirm. Hier schafft die Smalltalk-Klasse `BitBlt` Abhilfe.

BitBlts

Die Smalltalk-Klasse `BitBlt`[6] wird zum Kopieren von Pixeln aus einem Bild (`Form`) in ein anderes verwendet. Dafür wird zuerst ein

6 Abk. für Bit Block transfer

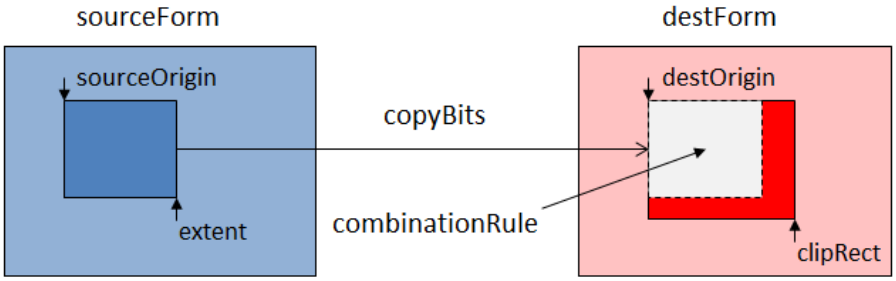

Abbildung 5: Bilder kopieren mit Hilfe der Klasse BitBlt

Objekt von dieser Klasse erzeugt (Listing 3). Hierbei müssen mehrere Parameter spezifiziert werden. Der erste Parameter (`destForm`) bestimmt das `Form`-Objekt, in dessen Bild die Pixel des zweiten `Form`-Objekts (`sourceForm`) kopiert werden. Mit `sourceOrigin` und `extent` wird ein Bereich aus dem ersten Bild festgelegt, der in den durch `destOrigin` und `clipRect` abgegrenzten Bereich des Zielbildes übertragen werden soll. Wird der Zielbereich kleiner als der zu kopierende gewählt, dann wird das Bild abgeschnitten. Wie die Pixel aus beiden Bereichen beim Kopieren kombiniert werden, z.B. Mischen oder Überschreiben, bestimmt der Wert des Parameters `combinationRule`. Alle möglichen Kombinationsvarianten sind in der Dokumentation der Klasse `BitBlt` zu finden. In Abbildung 5 ist das Zusammenwirken aller Parameter visualisiert.

```
1  form  := Form fromFileNamed: 'resouce.images\tachometer.png'
2  bitBlt := BitBlt destForm: Display
3                   sourceForm: form
4                   fillColor: nil
5                   combinationRule: (Form blend)
6                   destOrigin: 0@0
7                   sourceOrigin: 0@0
8                   extent: 40@40
9                   clipRect: (Display computeBoundingBox).
10 bitBlt copyBits.
```

Listing 3: Anzeigen von Bildern mit Hilfe der Klasse BitBlt

Das tatsächliche Kopieren der Bildpixel findet erst durch den Aufruf der Methode `copyBits` statt. Durch die Übergabe der globalen Smalltalk-Variable `Display` als `destForm`-Parameter, wie es in Listing 3 der Fall ist, wird das Anzeigen eines Bildes auf dem Bildschirm realisiert.

2.2 INTEGRATION NEUER AUTOMATIKEN IN AAF UND AAFGT

Dieser Abschnitt soll als eine kurze Anleitung für die Implementierung neuer Automatikfunktionen in AAF und deren Integration in AAFGT dienen. Für eine ausführlichere Beschreibung wird der inter-

essierte Leser auf die Arbeiten von Fuhrmann (2010, Anhang F) und Kosjar (2011) verwiesen.

Implementieren neuer Automatiken im AAF

Eine neue Automatik oder Automatikfunktion wird im AAF als eine Klasse realisiert, die vom `AAFAgent` ableitet. Aus diesem Grund werden Automatikfunktionen im ATEO-Projekt auch als *Agenten* bezeichnet. Ein Agent muss folgende Methoden implementieren:

- `readyForUse`
- `plaintextName`
- `compute: aSamState`
- `getAllProps` und `setAllProps: aPropDictionary`.

Die erste Methode zeigt die Einsatzbereitschaft des Agenten an, d.h. wenn die Methode `true` als Wert zurückgibt, dann wird der Agent im AAFGT als Element aufgelistet und kann beim Zusammenstellen von Automatikgraphen verwendet werden. Um das angezeigte Element mit einem Namen zu versehen, wird der Rückgabewert der Methode `plaintextName` verwendet. Dieser Name sollte eindeutig sein, damit das Element von allen anderen unterschieden werden kann. In der Methode `compute: aSamState` wird schließlich die Automatikfunktion implementiert. Der Parameter `aSamState` ist ein Objekt vom Typ `SamState` und enthält alle Informationen über den Zustand von SAM, die für Automatiken relevant sind. Durch die Veränderung dieses Zustands übt der Agent Einfluss auf SAM aus. Das übergebene Zustandsobjekt, mit oder ohne Änderungen seitens des Agenten, muss am Ende der Methode wieder zurückgegeben werden. Dadurch wird dieser Zustand von einem Agenten zum nächsten entlang eines Automatikgraphen weitergereicht, damit es am Ende wieder in SAM eingespeist werden kann.

Die letzten zwei Methoden müssen nur dann implementiert werden, wenn das Verhalten des Agenten durch Ändern von Parametern konfigurierbar ist. Wie ein Agent im AAFGT konfigurierbar gemacht wird, wird im nächsten Punkt dieses Abschnitts erläutert. Nachdem die Agenten in einer Automatik fertig konfiguriert wurden, kann ihr Graph und Zustand in eine Extensible Markup Language (XML)-Datei[7] persistent gespeichert werden. Hierbei wird zum Schreiben der Parameter jedes Agenten seine Methode `getAllProps` aufgerufen. Wird die Automatik zu einem späteren Zeitpunkt wieder geladen oder geöffnet, dann werden die Parameter ihrer Agenten durch `setAllProps:` auf die gespeicherten Werte gesetzt, womit der Zustand der Automatik wiederhergestellt wird.

7 Die Beschreibung des Formats der Datei ist in Fuhrmann (2010, Abschnitt 4.6) zu finden.

Integration neuer Automatiken ins AAFGT

Für die Konfiguration eines Agenten im AAFGT muss ein passender Dialog erstellt werden. Hierfür wird eine neue Klasse angelegt, die von `AAFAgentDialog` ableitet und denselben Namen hat wie der zugehörige Agent ergänzt mit der Endung "Dialog". Durch die Implementierung der Methode `addSpecialElements` kann der initiale Dialog mit den gewünschten Elementen erweitert werden. Da AAFGT in Morphic realisiert ist, müssen die neuen Dialogelemente auch mit Morphs implementiert werden. Um diese Aufgabe zu erleichtern sowie das Aussehen der Bedienelemente aller Agent-Dialogs zu vereinheitlichen hat Kosjar (2011, Abschnitt 5.2) in Squeak die *AAFGT Widget Gallery* implementiert. Hier kann sich der Programmierer einer großer Anzahl von bekannten grafischen Elementen wie Auswahlboxen, Knöpfe, Textfelder u.ä. bedienen. Jeder Dialog kriegt bei seiner Initialisierung automatisch durch AAFGT den zugehörigen Agenten in der Instanzvariable `delegate` zugewiesen. Über seine Methoden werden sowohl die anzuzeigenden Werte abgefragt als auch die neuen gesetzt.

2.3 SAM

In diesem Abschnitt werden neben der Versuchskonfiguration in SAM auch die Eigenheiten und die Erweiterungen der Versuchsumgebung vorgestellt, die für die Entwicklung und für den Einsatz von Automatiken relevant sind.

2.3.1 *Versuchskonfiguration*

Ein SAM-*Versuch* kann aus einem oder mehreren *Versuchsschritten* bestehen. Ein Versuchsschritt stellt dabei das Fahren einer bestimmten Strecke vom Start bis zum Ziel. Alle Versuchsschritte für einen Versuch werden in die Konfigurationsdatei von SAM *steps.txt* eingetragen. Ein Beispiel dieser Datei ist in Abbildung 6 zu sehen.

```
 7 # Syntax:
 8 #   step-number ';' mwi-control ';' track ';' obstacle-config [ ';' agent-config ]
 9
10  1;  1; lernstrecke;   obstacleConfig_0; test
11  2;  1; testabschnitt; obstacleConfig_0
12  3;  2; lernstrecke;   obstacleConfig_0
13  4;  2; testabschnitt; obstacleConfig_0
14  5; 12; hauptabschnitt_ohne_manipulationen; obstacleConfig_0
15  6; 12; hauptabschnitt_ohne_manipulationen; obstacleConfig_0
16  7; 12; hauptabschnitt1; obstacleConfig_1
17  8; 12; hauptabschnitt2; obstacleConfig_2
18  9; 12; hauptabschnitt_lang; obstacleConfig_7
19 10; 12; hauptabschnitt_lang; obstacleConfig_7
20 11; 12; hauptabschnitt_lang; obstacleConfig_7
```

Abbildung 6: SAM Konfigurationsdatei *steps.txt*

Die Zeilen, die mit einer Raute '#' beginnen, sind Kommentare. Jede der restlichen Zeilen, in denen die einzelnen Elemente durch ein Semikolon ';' getrennt sind, stehen für einen Versuchsschritt. Eine solche Versuchsschrittzeile enthält wie im Kommentar am Anfang der Abbildung zu sehen, die Versuchsschrittnummer, die Nummer der aktiven MWB (1, 2 oder 12), den Namen der Streckenkonfigurationsdatei und den Namen der Hinderniskonfigurationsdatei für die Strecke. Wenn für einen Versuchsschritt eine Automatik aktiviert werden soll, dann wird als fünftes Element in der Zeile der Name der Automatikdatei angegeben. Das ist die Datei, die den Graphen einer im AAFGT erstellten Automatik enthält. Alle Dateinamen in der Konfigurationsdatei werden ohne Dateinamenserweiterungen eingetragen.

2.3.2 *Ausführung von Automatiken*

Die Simulation einer Streckenfahrt (Versuchsschritt) in SAM wird durch die Ausführung von Simulationsschritten realisiert. Ein solcher Schritt, in SAM Tick genannt, sollte idealerweise 39ms dauern. Ist für den aktuellen Versuchsschritt eine Automatik vorgesehen, dann finden in einem Tick folgende Operationen[8] statt (in Abbildung 7 als Flussdiagramm dargestellt):

1. Joystick-Eingaben der MWB einlesen.
2. Für die Ausführung der Automatik das AAF aufrufen, `modelData`[9] als Parameter übergeben.
3. Aus den Joystick-Eingaben die neue Objektgeschwindigkeit und -richtung berechnen.
4. Aktuellen Streckenabschnitt zeichnen.
5. Hindernisse (wenn vorhanden) bewegen.
6. Trackingobjekt zu neuer Position bewegen.
7. Auf Hinderniskollision prüfen (Falls Kollision, Objekt neben Strecke setzen und 1s pausieren).
8. Die relevanten Daten für den Tick in `modelData` loggen.
9. Wenn Strecke nicht zu Ende bei Punkt 1 neu anfangen, sonst die Log-Einträge aus `modelData` in die Log-Datei schreiben und mit dem nächsten Versuchsschritt fortfahren.

8 In der Hauptschleife der Methode `processStep` von `SAMControllerStep`

9 Instanzvariable vom Typ `SAMModelData`, die den kompletten Zustand von SAM enthält. Weil AAF nicht alle Informationen braucht, werden nur die relevanten Daten als ein Objekt vom Typ `SamState` übergeben.

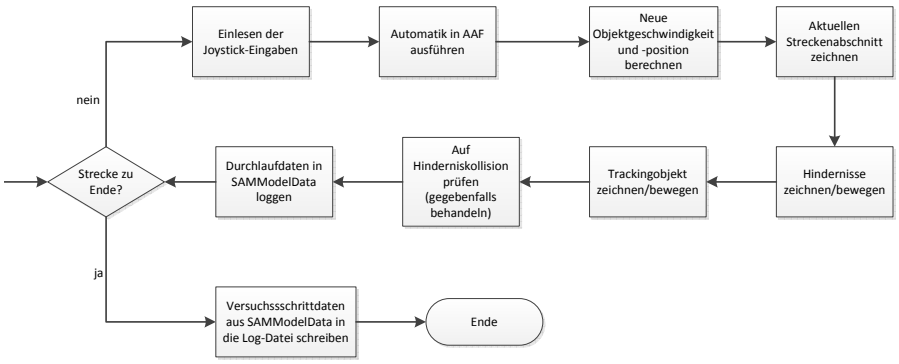

Abbildung 7: SAM Simulationsschritt

Jede der Operationen in einem Tick erzeugt Daten, die in `modelData` zwischengespeichert werden oder benutzt Daten, die aus `modelData` kommen. Dies und die oben beschriebene Reihenfolge von Operationen bringen aus der Sicht von Automatiken zwei Probleme mit sich, welche im Folgenden kurz dargestellt werden.

Alter SAM-Zustand für AAF

Das Problem hier kommt nur bei Automatiken vor, die die genaue Position des Trackingobjekts brauchen. Z.B. eine Automatik, die einen visuellen Hinweis am oder im Objekt anzeigt. Die Automatik bekommt zwar ein `SamState`-Objekt (Schritt 2), welches die Position des Trackingobjektst enthält, aber diese ist vom letzten Tick und somit nicht aktuell. Die neue Objektposition wird erst im nächsten Schritt nach der Ausführung der Automatik berechnet und in `modelData` aktualisiert. Benutzt die Automatik die übergebene, alte Position, so erscheint ihre Anzeige nicht an der gewünschten Stelle, es sei denn das Objekt hat seine Position nicht geändert.

Als Lösung für das Problem wurden in der Klasse `AAFValues`, Kategorie[10] *AAF-Agents-Concepts-Support* die Klassenmethoden

- `getTrackingObjXPosForCurrentState: aSamState`
- `getTrackingObjYPosForCurrentState: aSamState`

implementiert. Die Implementierung dieser Methoden besteht darin, den entsprechenden Code aus der Methode `processInputData` (berechnet die vertikale und horizontale Objektgeschwindigkeit) der Klasse `SAMControllerInput` zu kopieren. Wichtig dabei ist, dass die obigen Methoden im Unterschied zu der Methode `processInputData` die errechneten Werte nicht in `modelData` aktualisieren. Die Automatik kann jetzt mit Hilfe dieser Methoden die aktuelle Objektposition, die eigentlich erst nach der Ausführung der Automatik zur Verfügung steht, berechnen lassen und sie für ihre Anzeige benutzen.

10 In Squeak werden Klassen und Methoden einer Klasse für eine bessere Übersicht in Kategorien gruppiert.

Grafische Ausgaben von Automatiken

Das zweite Problem betrifft Automatiken, die jegliche Art von visuellen Ausgaben vorsehen. Würde die Automatik ihre Anzeige bei ihrer Ausführung (Schritt 2) anzeigen, so wäre diese für die MWB nicht sichtbar. Bei der Aktualisierung der Strecke (Schritt 4) wird der gesamte Bildschirmbereich, in dem die Strecke dargestellt wird, komplett gelöscht und neu gezeichnet. Dabei wird die schon produzierte Anzeige der Automatik, falls sie innerhalb des Streckenbereichs erscheint, ebenfalls gelöscht.

Das Problem wurde gelöst indem die Automatik ihre Anzeige nicht mehr selber anzeigt, sondern diese in `modelData` reinschreibt und an SAM weiterleitet. Für diesen Zweck wurden der Klassen `SAMModelData` die Instanzvariablen `bitBlts` und `postObjBlts` hinzugefügt. Beide sind vom Typ `OrderedCollection`. Wenn die Automatik etwas anzeigen will, dann muss sie ihre Anzeige in Form von einem oder mehreren `BitBlt`-Objekten in die `bitBlts`- oder `postObjBlts`-Variable von `modelData` oder in beide schreiben. Die Objekte aus der ersten Variable werden nach der Aktualisierung der Strecke aber vor dem Zeichnen des Objekts angezeigt, während diese aus der zweiten erst nach dem Zeichen des Objekts angezeigt werden. Damit können Anzeigen, die zum Teil vom Trackingobjekt überdeckt werden aber auch solche, die zum Teil oder ganz das Trackingobjekt überdecken, realisiert werden. Listing 4 zeigt die Methode `moveTrack` der Klasse `SAMViewTrack`, wo nach der Aktualisierung der Strecke (Zeilen 2-3) das Anzeigen der Objekte aus der Variable `bitBlts` stattfindet (Zeile 6).

```
1 moveTrack
2   track sourceOrigin: (0 @ (modelData trackHeight - modelData
        totalStepDistance)).
3   track copyBits.
4
5   (modelData bitBlts) do: [ :blt | blt copyBits].
6   modelData bitBlts: OrderedCollection new.
```

Listing 4: SAMViewTrack: Methode `moveTrack`

Die Variable `postObjBlts` wird auf dieselbe Art und Weise behandelt. Das geschieht allerdings in der Methode `moveTrackingObject` der Klasse `SAMViewTrackingObject`.

3 ANALYSE DER KONZEPTBÖGEN

In diesem Kapitel werden die Konzeptbögen analysiert. Diese enthalten die Konzepte, die von Entwicklern aus unterschiedlichen Bereichen wie Fahrzeugbau, Luftfahrt, Softwareentwicklung sowie von Teams aus Universitäten erarbeitet wurden. Insgesamt haben am Entwicklungsprozess 30 Teams, bestehend aus je drei Personen, teilgenommen. Die Konzeptbögen, die von den Entwicklern auf Papier geschrieben und größtenteils mit Skizzen ergänzt worden sind, wurden im Vorfeld dieser Arbeit von und im Auftrag von Kain digitalisiert. Die Analyse in dieser Arbeit basiert auf der digitalisierten Form der Bögen und der Skizzen.

Die folgenden Schritte beschreiben kurz den Entwicklungsprozess[1]:

- SAM und ihre Aufgabe wird den Teams vorgestellt.
- Jedes Team absolviert eine Probefahrt in SAM, nachdem es die Verhaltensspezifikation von SAM gelesen und sich ein Video von SAM angeschaut hat.
- Das Team stellt Fragen an exemplarischen MWB (Versuchsleiter selbst).
- Das Team bekommt ein Video von OA zu sehen.
- Fragen an exemplarischen Operateur (wieder Versuchsleiter selbst).
- Die Entwickler bekommen die Aufgabe, eine oder mehrere Automatiken zur Unterstützung und/oder Führung der MWB zu konzipieren. Dabei kann jede Automatik aus einer oder mehreren Funktionen bestehen. Das Ziel der Automatiken ist, den Operateur zu ersetzen.
- Die Ergebnisse werden von jedem Team schriftlich auf einem Konzeptbogen dokumentiert. Zum besseren Verständnis der Konzepte können zusätzlich Skizzen erstellt werden.
- Der gesamte Prozess dauert mehrere Stunden, wobei maximal 2 Stunden für die Erstellung der Konzepte anfallen, und wird als Video aufgezeichnet.

Insgesamt sind dadurch 114 Automatiken und 279 Funktionen entstanden. In den folgenden Abschnitten werden die Analyse von diesen Konzepten und ihre Ergebnisse vorgestellt.

1 Kain (2012, in Vorb.) beschreibt den Prozess im Detail.

3.1 ANALYSE

Für die Analyse wurden die Konzepte von allen 30 Teams auf Automatiken und Automatikfunktionen untersucht, die eine visuelle Darstellung der aktuellen und optimalen Objektposition, -richtung und -geschwindigkeit und das Anzeigen der aktuellen und optimalen Joystickposition der MWB beinhalten. Die Funktionen mit allen anderen Arten von visuellen Anzeigen oder Hinweisen werden in der Arbeit von Kosjar (2012) bearbeitet. Eine besondere Behandlung wird dabei für die Konzepte erfordert, die Anzeigen für Hindernisse auf der Strecke vorsehen. Hinweise, die das Aufkommen von statischen oder dynamischen Hindernissen signalisieren, sowie Anzeigen zum Umfahren von statischen Hindernissen gehören ebenfalls zur Arbeit von Kosjar. Visuelle Hinweise zur Vermeidung von Objektkollisionen mit dynamischen Hindernissen dagegen werden in dieser Arbeit behandelt. Der Grund dafür ist, dass die letztgenannte Art von Hinweisen entweder eine Erhöhung oder eine Verringerung der Objektgeschwindigkeit anzeigt und damit als visuelle Darstellung der optimalen Objektgeschwindigkeit gilt.

Die Teams hatten für das Entwickeln und das Aufschreiben der Konzepte nur wenige Stunden, weshalb viele der enthaltenen Automatikfunktionen ungenau und nicht ausreichend detailliert spezifiziert worden sind. Das erschwert die Umsetzung der Konzepte. Um dieses Problem zu lösen und eine Realisierung maximal vieler Automatikfunktionen zu erreichen, wurden während der Analysephase der Konzepte wöchentlich mit allen zu diesem Zeitpunkt am ATEO-Projekt beteiligten Personen - Saskia Kain, Nicolas Niestroj, Nikolai Kosjar, Andreas Wickert, Helmut Weidner-Kim und Aydan Seid, Treffen organisiert. Das Ziel dieser Treffen bestand darin die ungenauen Stellen in den Funktionsbeschreibungen genau zu spezifizieren. Die Vorgehensweise hierfür war, sich die Beschreibung der anderen Funktionen des jeweiligen Teams und eventuell die dazugehörige Videoaufzeichnung anzuschauen und dadurch versuchen die Spezifikation der Funktion sinnvoll zu vervollständigen. Die Implementierung von einigen Automatiken, die im Rahmen dieser Arbeit realisiert wurden, entsprechen deshalb nicht immer eins zu eins ihrer Beschreibung aus den Konzeptbögen.

3.2 ERGEBNISSE

Die Ergebnisse der Analyse der Konzeptbögen, d.h. die in dieser Arbeit zu implementierenden Automatiken und deren Funktionen, sind in Tabelle 1 aufgelistet. In jeder Zeile der Tabelle sind die Teamnummer des Entwicklerteams, die Namen der Automatiken (mit fetter Schriftstärke) sowie die Namen ihrer Funktionen (eingerückt) enthalten. Diese wurden aus den Originalkonzeptbögen übernommen.

Tabelle 1: Zu implementierende Automatiken

Team[2]	Automatiken und Funktionen
20	**Verhindern des Verlassens der Strecke** Anzeige der optimale Route/Spur
21	**Anzeige** Anzeige
22	**Geschwindigkeitsassitenz/Kollisionsvermeidung** Brake cue Speed cue
23	**Joystick der anderen Person anzeigen** Normal Master **Handlungsanweisungen** Anzeige
24	**Ideallinienanzeige + Sollgeschwindigkeit** Ideallinienanzeige + Sollgeschwindigkeit **Farbänderung Kreisfläche** Farbänderung Kreisfläche
26	**Anzeige Richtungspfeil** Anzeige Richtung der Bewegung Objektgeschwindigkeit Visualisierung der optimalen Objektbewegung
28	**Anzeige** Ist-Geschwindigkeitsanzeige Geschwindigkeitsempfehlung
29	**Anzeige** Alarm Toleranzbereich
30	**Geschwindigkeitsregulierung** Überschreitung der optimalen Geschwindigkeit Unterschreitung der optimalen Geschwindigkeit
32	**Überwachung der Ideallinie** Visuelle Anzeige der Ideallinie **Tempomat** Visuelle Anzeige Gas geben oder Bremsen
33	**Sollvorgabensteuerung** Sollvorgabenanzeigem

2 Die 30 Teams sind von 20 bis 49 durchnummeriert.

34	**Ausgabe der Führungsgrößen** Ausgabe der optimalen Steuerungssignale
35	**Geschwindigkeitskontrollanzeige** Anzeige Signal
38	**Steering Assistanz - Evaluation & Selection** Display proposal command **Steering Assistanz - Command Generator** Speed advisor dynamic obstacle
39	**Assistenz** Anzeige der optimalen Linie Anzeige der optimalen Steuerungskommandos
41	**Optimizer** Informer
42	**Warnung bei potentiellen Kollisionen** Kollision Warnung visuell
45	**Entscheidungsassistent** Anzeige (bei dynamischen Hindernissen)
46	**XY-Langi-Lati-Control** Geschwindigkeitsinformation
48	**Steuerungsassistenz** Steuerungsanzeige
49	**Navigationshilfe** Steuerungsdisplay

Wie in Tabelle 1 zu sehen ist, haben etwa zwei Drittel der Teams in ihren Konzepten das Anzeigen der optimalen und aktuellen Joystick- und Objektposition vorgesehen. Die Gründe für diese Art von Anzeigen sind zwei. Im Allgemeinen dienen diese der Unterstützung der MWB, d.h. das Erscheinen der Hinweise oder der Anzeigen und ihre Befolgung oder Wahrnehmung durch die Versuchspersonen soll ihnen helfen die Trackingaufgabe möglichst schnell und genau zu absolvieren. Der speziellere Grund ist das Schaffen eines Kommunikationskanals zwischen den MWBn. Wie bereits im Abschnitt 1.1 erwähnt wurde, dürfen die Versuchspersonen während einer Fahrt nicht miteinander kommunizieren. Durch die visuelle Darstellung der aktuellen und/oder optimalen Position des Trackingobjekts oder der Joysticks der MWB können die beiden Personen die Absichten des Partners erahnen und dadurch mögliche Konfliktsituationen schneller entschärfen.

Um die in der Tabelle 1 aufgelisteten Automatikkonzepte und deren Implementierung kompakter und übersichtlicher beschreiben zu

Tabelle 2: Klassifikation der Konzepte

#	Klassenname und Teams
1	**Separate Anzeige der Steuerung(svorgaben)** 21, 23, 33, 34, 35, 48, 49
2	**Gemeinsame Anzeige der Steuerung(svorgaben)** 30, 38
3	**Anzeige am/im Trackingobjekt** 20, 23, 24 (2), 26, 28, 32, 39
4	**Visuelle Hinweise zur Vermeidung von Kollisionen mit dynamischen Hindernissen** 22, 38, 41, 42, 45, 46, 49
5	**Anzeige der Ideal- und Mittellinie** 29, 32, 33, 39, 48

können, wurde der Versuch unternommen, diese Konzepte nach der Art ihrer visuellen Anzeige zu klassifizieren. Das Ergebnis der Klassifizierung ist in Tabelle 2 zu sehen. Der Name der Klassifikationsklasse ist in fetter Schrift angegeben. Die Nummern der Teams, die eine der Klasse entsprechende Anzeige enthalten sind darunter eingerückt aufgelistet. Manche Teams haben Automatikfunktionen für mehr als eine Klasse konzipiert, deshalb kommt ihre Nummer in mehreren Tabellenzeilen vor. Eine *(2)* hinter der Teamnummer bedeutet, dass das Team mit zwei Funktionen in der angegebenen Klasse vertreten ist.

Im Folgenden erfolgt eine kurze Beschreibung der Klassen aus der Konzeptklassifikation aus Tabelle 2. Die Anzeigen der Teams aus den Klassen 1, 2, 3 und 5 sind während der Fahrt einer Strecke durchweg aktiv, d.h. die Anzeige ist vom Start bis zum Ziel zu sehen. Die Anzeigen der Automatikfunktionen aus Klasse 4 dagegen werden nur im Falle von einer vorauserkannten Kollision mit einem dynamischen Hindernis in Form eines Hinweises zur Erhöhung oder zur Verringerung der Objektgeschwindigkeit eingeblendet.

Separate Anzeige der Steuerung(svorgaben)

Diese Klasse von Anzeigen sieht eine getrennte Anzeige für jeden der beiden MWB vor. Diese sind normalerweise links und rechts auf der Strecke auf der Höhe des Trackingobjekts positioniert. Bei allen sieben Anzeigen in dieser Klasse wird die Joystickauslenkung der Versuchspersonen visualisiert. Dabei können auf jeder der beiden Anzeigen folgende Kombinationen von aktueller und optimaler Joystickposition auftreten

- eigene aktuelle und optimale Position,

- nur eigene optimale Position,
- Differenz von eigener aktueller und optimaler Position,
- Differenz von eigener aktueller vertikaler und optimaler vertikaler Position,
- eigene aktuelle und aktuelle Position des Partners und die Summe beider Positionen.

Die einzelnen Positionen werden in Form eine Pfeils, eine durchgängige oder gestrichelte Linie, eine Linie mit einem Kreis an der Spitze oder nur als Kreise auf einem farbigen oder transparenten Hintergrundbereich gezeichnet.

Gemeinsame Anzeige der Steuerung(svorgaben)

In dieser Klasse sind Anzeigen enthalten, die gemeinsam für die beiden MWB sind, d.h. die Information in der Anzeige ist für beide Personen relevant und gleich. Es wird ausschließlich die optimale Joystickposition für die MWB angezeigt. Im Unterschied zu den Anzeigen mit optimaler Position aus der vorigen Klasse, bezieht sich die hier angezeigte optimale Position auf die Summe der optimalen Positionen beider Joysticks. Da jeder der MWB mit einem bestimmten Anteil[3] an der Gesamtsteuerung beteiligt ist, müssen beide ihre Joysticks bewegen um die optimale gemeinsame Position zu erreichen.

Anzeige am/im Trackingobjekt

Die Konzepte die eine Anzeige am/im Trackingobjekt vorsehen, bilden die dritte Klasse von Anzeigen aus Tabelle 2. Wie der Name es vermuten lässt, bewegen sich die Anzeigen in dieser Klasse immer mit dem Objekt zusammen. Bis auf die Anzeige des letzten Teams, die die gemeinsame optimale Position für die Joysticks beider MWB in Form eines ausgefüllten Kreises innerhalb des Objekts visualisiert, findet man in allen anderen Anzeigen der Klasse die Darstellung der optimalen Objektrichtung oder -geschwindigkeit oder beides.

Für die Darstellung der optimalen Objektrichtung wird in allen relevanten Konzepten ein Pfeil verwendet, der am Objekt gezeigt wird. Durch die Färbung dieses Pfeils oder durch das Ändern seiner Länge wird die Anzeige der optimalen Geschwindigkeit realisiert. Es gibt auch Teams, die keinen Pfeil in ihrer Anzeige vorsehen. Das Anzeigen der optimalen Objektgeschwindigkeit geschieht in diesem Fall durch eine Färbung des Objekts. Je nachdem wie viel seine Geschwindigkeit von der optimalen abweicht, wird dieses meistens in den folgenden drei Farben gefärbt:

3 Vorgegeben sind 50%, aber der Operateur oder eine aktive Automatik mit harten Eingriffen kann diesen Anteil auf einen Wert zwischen 0 und 100 ändern.

- grün - sehr geringe Abweichung,
- gelb oder orange - Abweichung innerhalb eines konfigurierbaren Toleranzbereichs,
- rot - Abweichung außerhalb des Toleranzbereichs.

Visuelle Hinweise zur Vermeidung von Kollisionen mit dynamischen Hindernissen

Eine Kollision des Trackingobjekts mit einem dynamischen Hindernis bedeutet sowohl Zeitfehler als auch Flächenfehler für die Versuchspersonen und führt somit zu einer schlechteren Bewertung des jeweiligen Teams. Um dies zu verhindern, haben viele Entwickler-Teams für diese Situation einen visuellen Hinweis vorgesehen. Dieser fordert die MWB, in Form eines Pfeils oder einer Textnachricht, entweder die Objektgeschwindigkeit zu erhöhen oder diese zu verringern um eine Kollision zu vermeiden. Die Berechnung der Kollisionswahrscheinlichkeit wird durch das Eintreten eines dynamischen Hindernisses in den sichtbaren Streckenbereich ausgelöst und wird bis zu seinem Verschwinden für jeden Tick neu berechnet.

Anzeige der Ideal- und Mittellinie

Die letzte Klasse aus Tabelle 2 enthält Anzeigen, die entweder die Mittellinie jeder in SAM zu fahrende Strecke oder die Ideallinie, mit der die Versuchspersonen die jeweilige Strecke am schnellsten fahren können, farblich hervorheben oder diese in Form einer anderen Anzeige darstellen. An dieser Stelle sei darauf hingewiesen, dass nicht die Ideallinie, sondern die Mittellinie in SAM die optimale ist. Der Grund dafür liegt in der Methode für die Berechnung des Flächenfehlers. In dieser wird jeder Abweichung des Objektmittelpunkts von der Mittellinie als Fehler bewertet. Da die Ideallinie in Kurven mit der Mittellinie nicht übereinstimmt, ist diese nicht optimal.

4 PRAKTISCHE UMSETZUNG

In diesem Kapitel wird die Implementierung der Automatikfunktionen aus Tabelle 1 beschrieben. Die Konzepte für diese wurden im Kapitel 3 analysiert und nach der Art ihrer Anzeige klassifiziert (Tabelle 2). Die Beschreibung in diesem Kapitel basiert auf die Klassifikationsklassen aus dieser Tabelle und der Name jedes der folgenden Abschnitte entspricht einem Klassennamen. An dieser Stelle sei darauf hingewiesen, dass die Automatikfunktionen in manchen Klassen in mehr als einer Smalltalk-Klasse implementiert wurden.

Bei der Implementierung der Anzeigen wurden fertige Bilder benutzt. Diese wurden in Anlehnung an die Skizzen, die die Entwickler bei der Erstellung der Konzepte in jeweiligem Team gezeichnet haben, vom Autor selbst angefertigt. Nur bei der Zeichnung von geraden, dynamischen Linien oder mit einer Farbe gefüllten Kreisen und Vierecken wurden built-in Squeak-Funktionen verwendet. Die Klassen, in denen die Automatikfunktionen in dieser Arbeit implementiert wurden, sind in Squeak in der Kategorie *AAF-Agents-Concepts-Visual* zu finden. Zu jeder Automatikklasse wurde außerdem ein Konfigurationsdialog entwickelt, dessen Klassen (eine für die Konfigurationselemente und eine für die Vorschau der Automatikausgabe) sich in den Kategorien *AAFGT-Dialog* und *AAFGT-Widgets-Agents* befinden. Die Klassen aus den letztgenannten zwei Kategorien haben denselben Namen wie die Automatikklasse erweitert mit der Zeichenkette "Dialog" bzw. "PreviewMorph". Die komplette Liste mit den Konfigurationsparametern für jede implementierte Automatikklasse ist in Anhang C zu finden.

4.1 SEPARATE ANZEIGE DER STEUERUNG(SVORGABEN)

In diesem Abschnitt wird die Implementierung der Automatikfunktionen vorgestellt, die eine getrennte Anzeige für jeden der beiden MWB vorsehen. Tabelle 3 listet die Squeak-Klassen auf, die hierfür implementiert wurden. In der linken Spalte der Tabelle ist der Name der Klasse angegeben, unter welchem diese in Squeak gefunden werden kann, gefolgt vom Namen[1] der Klasse, mit dem sie in AAFGT gezeigt wird. Die zweite Spalte enthält die Nummern der Teams, wessen Automatikfunktionen in der Klasse links umgesetzt wurden.

In den folgenden Unterabschnitten wird die praktische Umsetzung der einzelnen Klassen detailliert beschrieben.

1 Diese wurden teilweise oder ganz aus den relevanten Konzepten übernommen oder vom Autor entsprechend der Anzeige der Klasse vergeben.

Tabelle 3: Implementierte Squeak-Klassen für separate Steuerungsanzeigen

Klasse	Team
`AAFJoystickNavigationDisplayAgent` "Joystick-Navigationsanzeige"	21, 23, 33, 48, 49
`AAFSpeedAndControlRegulationArrowsAgent` "Steuerungsregulierung"	34
`AAFSpeedRegulationArrowsAgent` "Geschwindigkeitsregulierung (Pfeil)"	35

4.1.1 *Joystick-Navigationsanzeige*

Wie in Tabelle 3 zu sehen ist, werden die meisten Automatikfunktionen, die eine getrennte Anzeige für die Versuchspersonen vorsehen, in der Squeak-Klasse `AAFJoystickNavigationDisplayAgent` implementiert.

In Abbildung 8 ist die Anzeige von Team 21 zu sehen. Gezeigt werden die Ist- und Soll-Position der Joysticks der beiden MWB. Dabei zeigt der schwarze Kreis die Soll-Position und der weiße Kreis die Ist-Position für den jeweiligen Joystick. Als Hintergrund dienen die schwarz umrandeten Quadranten eines grau gefüllten Kreises. Die Koordinaten für die Joystick-Ist-Positionen werden aus `SAMModelData` entnommen. Diese Werte entsprechen der Joystickeingaben, die die beiden Versuchspersonen über ihre Joysticks vornehmen. Die Berechnung der Soll-Position für die beiden Joysticks wird im Rahmen der Studien- und Diplomarbeit von Weidner-Kim (2012, in Vorb.) behandelt und implementiert. Auf diese Werte wird mit Hilfe der Methoden

- `optimalFromX: x Y: y`
- `optimalForkFromX: x Y: y branch: b`

der Klasse `AAFOptimalStep` aus der Kategorie *AAF-Agents-Concepts-Support* zugegriffen. Die erste Methode wird immer dann aufgerufen, wenn auf der Strecke keine Gabelung liegt, wahrend die zweite nur im Falle einer Gabelung angewendet wird (Ob eine Gabelung vorliegt oder nicht, wird mit Hilfe der Variable `distanceDictionary` in `SAMModelData` bestimmt). Die oben aufgelisteten Methoden erwarten als Parameter die aktuelle `x`- und `y`-Position des Trackingobjektes. Die zweite Methode erwartet außerdem einen dritten Parameter, der nur die Zeichenketten `#left`, `#right` oder `#both` als Wert haben darf. Abhängig von diesem Parameter berücksichtigt der Algorithmus bei der Berechnung der Joystick-Soll-Position entweder nur den linken oder nur den rechten oder beide Zweige der Gabelung. Diese zwei Methoden für die Bestimmung der Joystick-Soll-Position werden nicht nur

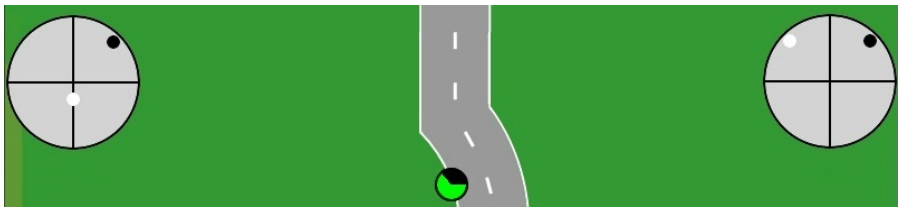

Abbildung 8: Team 21, Joystick Ist- und Soll-Position

von der Automatik der Anzeige in Abbildung 8 verwendet, sondern von allen Automatiken, die ein Anzeigen der Soll-Position (in welcher Form auch immer) vorsehen.

Die Werte für die Ist-Position der Joysticks liegen sowohl auf der X-Achse als auch auf der Y-Achse im Intervall [-1024;1024]. Dasselbe Intervall haben die Werte der Soll-Position. Da die Automatikanzeige, gemessen in Pixel, viel kleiner ist als dieses Intervall, werden diese Werte von der Automatik auf die Größe der Anzeige skaliert.

Alle grafischen Elemente der Anzeige wurden ausschließlich unter Verwendung von Standardklassen von Squeak realisiert.

```
1  refreshDisplays
2    | display tmpForm |
3    ((displayType isNil not) & (displayWidth isNil not) &
4    (displayBgColor isNil not) & (displayFrameColor isNil not))
5    ifTrue: [
6      mwiLeftDisplayForm := (Form extent:
7                               ((displayWidth @ displayWidth) +
8                               (self class circleExtent))
9                               depth: (Display depth)).
10     mwiLeftDisplayForm fillColor: (Color transparent).
11
12     (displayType = (self class noDisplay))
13     ifFalse: [
14       (displayType = (self class circleDisplay))
15         ifTrue: [
16           display := CircleMorph new.
17         ].
18
19       (displayType = (self class squareDisplay))
20         ifTrue: [
21           display := RectangleMorph new.
22         ].
23
24     display extent: (displayWidth @ displayWidth);
25     borderWidth: (self class defaultDisplayFrameWidth);
26     borderColor: displayFrameColor;
27     color: displayBgColor.
28     tmpForm := display imageForm.
29     ...
```

Listing 5: Erzeugen von Joystick-Anzeigen mit Squeak

Abbildung 9: Team 48, Joystick Ist- und Soll-Position

In Listing 5 ist ein Teil des Codes der Methode `refreshDisplays` zu sehen. Die Methode wird zum Erzeugen des Hintergrundbildes (falls konfiguriert) von der Anzeige verwendet. Ist ein Kreis oder ein Quadrat als Hintergrund gewünscht, dann wird dieser durch Instanziierung der Klasse `CircleMorph` bzw. `RectangleMorph` erzeugt. Das neue Objekt wird in den Zeile 24-27 mit den vom Anwender bestimmten Parametern konfiguriert und anschließend wird in Zeile 28 das fertige Bild zur Weiterverwendung aus dem Morph[2] in ein `Form`-Objekt gespeichert. Wenn kein Hintergrund gewünscht ist, dann macht die Methode nichts. Das Zeichnen der Punkte, die die optimale und aktuelle Joystickposition darstellen, geschieht analog.

Die Anzeige von Team 48 in Abbildung 9 enthält dieselbe Information wie die Anzeige von Team 21. Statt den weißen und schwarzen Kreisen werden hier Linien(Vektoren) für die Darstellung der aktuellen und optimalen Joystickposition benutzt. Eine rote und durchgezogene Linie für die Soll-Position, eine schwarze und gestrichelte für die Ist-Position. Der Hintergrund ist transparent. Zum Zeichnen der durchgezogenen und der gestrichelten Linie wurden die Methoden

- `line: fromPt to: toPt width: w color: c`
- `line: fromPt to: toPt width: w color: c1 dashLength: l1 secondColor: c2 secondDashLength: l2 startingOset: o`

verwendet. Diese werden von der Squeak-Klasse `Canvas` bereitgestellt. Die Benutzung der ersten Methode zum Zeichnen von durchgezogenen Linien in der Klasse `AAFJoystickNavigationDisplayAgent` ist in Listing 6 zu sehen.

```
1 drawSolidLineOn: displayForm to: toPoint withColor: aColor
2   | fromPoint |
3
4   fromPoint := (displayForm extent) / 2.
5   (displayForm getCanvas) line: fromPoint
6                           to: (fromPoint + toPoint)
7                           width: 2
8                           color: aColor.
```

Listing 6: Zeichnen von dynamischen Linien in Squeak

2 Wie bereits in Unterabschnitt 2.1.2 erwähnt wurde, sind Morph-Objekte zu langsam für die Anzeigen von Automatiken. Die Anzeigen werden deswegen mit Hilfe von `Forms` und `BitBlts` realisiert.

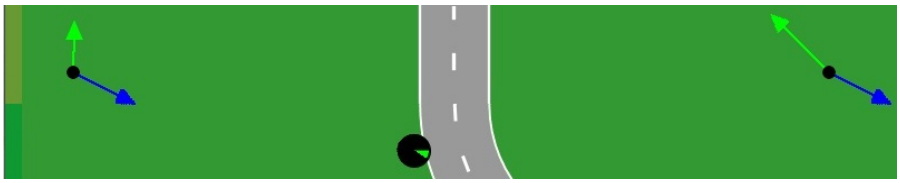

Abbildung 10: Team 49, Joystick Ist- und Soll-Position

Abbildung 11: Team 23, Joystickposition jedes MWBs und deren Summe

Wie die beiden bereits beschrieben Methoden zur Bestimmung der Soll-Position des Joysticks, werden auch diese beiden Methoden zum Zeichen von durchgezogenen und gestrichelten Linien von allen Automatiken verwendet, deren Anzeigen sich in der Richtung und in der Länge ändernde Linien enthalten.

Team 49 zeigt auch die Ist- und Soll-Position für den Joystick, allerdings in Form von Pfeilen (Abbildung 10). Der blaue Pfeil gibt die Soll-Postion an und der grüne die Ist- Position. Es gibt kein Hintergrundbild.

Die letzten beiden Anzeigen in diesem Unterabschnitt sind in Abbildung 11 und in Abbildung 12 zu sehen. Optisch betrachtet sehen die beiden Anzeigen sehr ähnlich aus. Beide haben als Hintergrund ein Rechteck, das durch zwei schwarze Linien horizontal und vertikal in vier gleichen Teilen geteilt ist und beide enthalten mindestens einen Vektor. Betrachtet man die Anzeigen konzeptuell, dann stellt man fest, dass sie sich gänzlich voneinander unterscheiden. In der Anzeige von Team 23 werden mit einem roten und gelben Vektor die momentane Joystickposition jedes MWBs und mit einem grünen Vektor die Summe der beiden Positionen angezeigt. Die Anzeige ist für beide MWB identisch. Die Anzeige von Team 33 sieht nur das Anzeigen der optimalen Joystickauslenkung vor. Diese wird in Form eines schwarzen Pfeils dargestellt.

Eine letzte Sache, die nur die Implementierung der Anzeigen in Abbildung 8 und Abbildung 9 betrifft, muss an dieser Stelle etwas

Abbildung 12: Team 33, Optimale Joystickposition für beide MWB

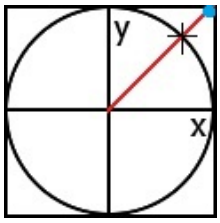

Abbildung 13: Rechteck und Kreis als Koordinatensysteme in SAM

genauer erläutert werden. Die zwei Konzepte, die hinter den Automatiken dieser Anzeigen stecken, sehen einen Kreis als Koordinatensystem für die Anzeige der Joystickpositionen vor, während SAM als Joystickeingaben die Koordinaten von einem rechteckigen Koordinatensystem liefert. Abbildung 13 soll das Problem veranschaulichen. Bei einem Vollausschlag des Joysticks auf der X- und Y-Achse liefert SAM die Koordinaten, die der Position des blauen Punktes in der Abbildung entsprechen. Zeichnet man diesen Punkt, anhand der von SAM gelieferten Werte, in dem kreisförmigen Koordinatensystem, das sich in dem rechteckigen Koordinatensystem befindet, dann befindet sich der blaue Punkt außerhalb des Kreises. Obwohl die Anzeige mit dem blauen Punkt außerhalb des Kreises an sich nicht falsch ist, kann sie irritierend und seltsam auf die MWB wirken. Damit der blaue Punkt auf dem Kreis und nicht außerhalb von ihm gezeichnet werden kann, wurde in der Automatikklasse die Methode `getIntersectionWithCircle: aPoint radius: radius`[3] implementiert. Diese berechnet den Schnittpunkt der roten Gerade, die die Joystickauslenkung darstellt, mit dem kreisförmigen Koordinatensystem. Das ist die Stelle an dem der blaue Punkt auf dem Kreis gezeichnet wird.

Konfigurationsdialog

Der Konfigurationsdialog von `AAFJoystickNavigationDisplayAgent` ist in Abbildung 14 visualisiert. Hier kann man unter anderem die Größe, die Farbe, die Position auf der Strecke und den anzuzeigenden Inhalt der Anzeige einstellen. Eine detaillierte Beschreibung aller Parameter und deren Werte ist im Abschnitt C.1 zu finden.

Die Elemente des Konfigurationsdialogs, wie z.B. Auswahl- und Checkboxen wurden mit Hilfe der Klassen aus der *AAFGT Widget Gallery* von Kosjar (2011, Abschnitt 5.2) realisiert. Dieser hat auch das sogenannte Ereignis-Model für SAM konzipiert und implementiert (Kosjar, 2012, Abschnitt 6.1). Es ist in der Graphical User Interface (GUI) für die Konfiguration von jeder Automatik integriert und wird benutzt, um zu bestimmen, beim Auftreten welcher Ereignis-

3 Die Implementierung der Methode basiert auf dem Code der Methode `intersectionWithLineSegmentFromCenterTo: aPoint` aus der Squeak-Klasse `EllipseMorph`.

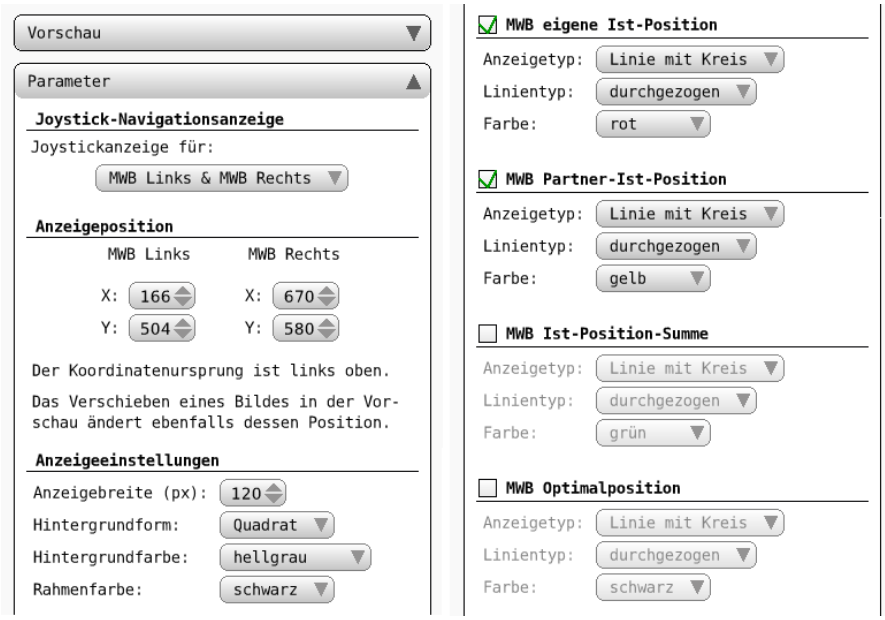

Abbildung 14: Konfiguration von Joystick-Navigationsanzeige

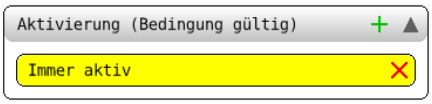

Abbildung 15: Ereignis für immer aktive Automatik

se die Automatikfunktion aktiviert werden muss. Alle in dieser Arbeit vorkommenden Automatiken, außer die Automatik aus Unterabschnitt 4.4, sind so vorkonfiguriert, dass ihre Anzeigen während der ganzen Fahrt aktiv sind (Abbildung 15).

Vorschau

Damit der Anwender die Änderungen in der Automatikanzeige bereits während ihrer Konfiguration sieht, wurde für jede Automatik eine Vorschau implementiert. Abbildung 16 zeigt diese für die aktuelle Automatik für die Parameter aus Abbildung 14.

Zum Ändern und zum Auslesen des Wertes von jedem Parameter durch den Konfigurationsdialog, werden sogenannte *setter*- und *getter*-Methoden benutzt.

```
displayBgColor: aColor

  (aColor ~= displayBgColor)
    ifTrue: [
      displayBgColor := aColor.
      self refreshDisplays.
      self agentPropertyChanged.
    ].
```

Listing 7: Änderung eines Automatikparameters

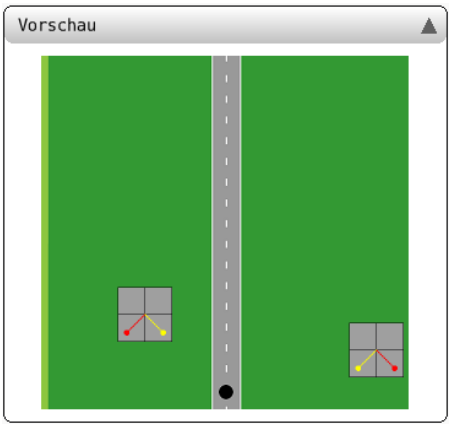

Abbildung 16: Vorschau von Joystick-Navigationsanzeige

Damit die Vorschau auf Änderungen von einem Parameterwert reagieren kann, müssen die *setter*-Methoden jede Änderung in Form eines Ereignisses verkünden (Listing 7, Zeile 7). Bemerkt die Vorschau ein solches, dann kann sie ihre Anzeige aktualisieren.

Eine Sache, die bei diesem Konfigurationsdialog zu sehen war und auch bei weiteren zu sehen sein wird, aber noch nicht angesprochen wurde, ist die Drag-and-Drop-Funktionalität der Anzeigeelemente in der Vorschau. Der Benutzer wird mit einem entsprechenden Text im Konfigurationsbereich, in dem die Positionierung der Anzeige vorgenommen werden kann, darauf hingewiesen. Um diese Funktionalität in die Vorschau einzubauen, muss die implementierende Klasse von der Squeak-Klasse `ImageMorph` ableiten und bei der Initialisierung die geerbte Methode `enableDragNDrop:` mit dem Parameter `true` aufrufen. Die Klasse `AAFGTDNDBoardMorph` in der Kategorie *AAFGT-Widgets-Agents* besitzt bereits diese und weitere nützlichen Eigenschaften, weshalb die Vorschau-Klasse am Besten diese erweitern sollte. Zusätzlich muss die Klasse die geerbte Methode

- `acceptDroppingMorph: aMorph event: evt`

überschreiben. Diese behandelt das "Droppen" von einem Anzeigeelement in der Vorschau. Das Anzeigeelement an sich ist auch ein Morph, welcher eine Instanz der Klasse `AAFGTDNDImageMorph` darstellt. Die letztgenannte Klasse ist ebenfalls eine Unterklasse von `ImageMorph`. Sie stellt die Funktionalität bereit, die zum Verschieben des Elements mit einer Maus benötigt wird.

4.1.2 *Steuerungsregulierung*

Die nächste Automatikklasse in diesem Abschnitt implementiert nur das Konzept vom Team 34. In Abbildung 17 ist die Anzeige der Klasse `AAFSpeedAndControlRegulationArrowsAgent` zu sehen. Diese ist auch eine Art Anzeige der Ist- und Soll-Position des Joysticks von

Abbildung 17: Team 34, Differenz von Joystick Soll- und Ist-Position

jedem MWB, wie die Automatik aus dem vorigen Unterabschnitt. Die beiden Größen werden in der Anzeige von dieser Automatik aber nicht explizit dargestellt, sondern nur ihre Differenz und zwar in Form eines schwarzen Pfeils. Stimmen die Ist- und Soll-Position eines Joysticks überein, dann wird kein Pfeil angezeigt. Sobald der Joystick eines MWBs von der Soll-Position abweicht, wird für die Korrektur dieser Abweichung nötige Joystickauslenkung als Pfeil visualisiert.

Der Teil der `compute`-Methode, der die Abweichung der Ist-Position von der Soll-Position berechnet ist in Listing 8 zu sehen. Dafür werden zuerst die X- und Y-Position des Trackingobjekts für den aktuellen Tick bestimmt (Zeile 8-9). Diese Werte werden dann benutzt, um die optimale Position des Objekts zu bestimmen, welche laut Weidner-Kim (2012, in Vorb.) auch der optimalen Joystickposition für die beiden MWB entspricht. Welche der beiden Methoden für die Berechnung der optimalen Objektposition (Zeile 12 oder 15) aufgerufen wird, bestimmt die Entfernung des Objekts von der nächsten Gabelung.

```
1  compute: aSamState
2    | nextFork newX newY optimalPosXY diffOptPosMwiLeft
         diffOptPosMwiRight |
3
4    "Check if there is a fork in front of us"
5    nextFork := distDict at: #nextFork.
6
7    "the optimal calculation needs the new positions of the
         tracking objects:"
8    newX := AAFValues getTrackingObjXPosForCurrentState: aSamState.
9    newY := AAFValues getTrackingObjYPosForCurrentState: aSamState.
10   ((nextFork isNil) or: [nextFork distance > (AAFValues
         forkLookAhead)])
11     ifTrue: [
12       optimalPosXY := optCalc optimalFromX: newX Y: newY.
13     ]
14     ifFalse: [
15       optimalPosXY := optCalc optimalForkFromX: newX Y: newY
             branch: #both.
16     ].
17
18   diffOptPosMwiLeft := optimalPosXY - (aSamState joystickRaw1).
19   diffOptPosMwiRight := optimalPosXY - (aSamState joystickRaw2).
```

Listing 8: Berechnung der Werte für die Steuerungsregulierung

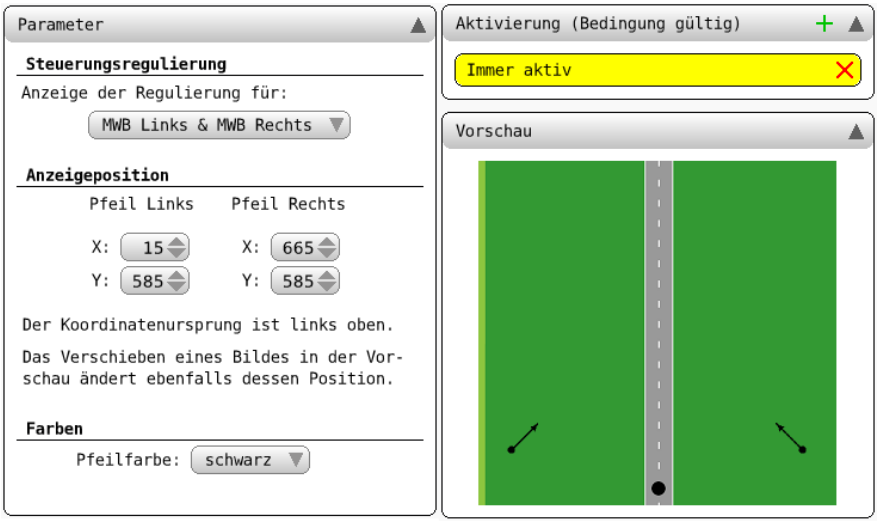

Abbildung 18: Steuerungsregulierung: Konfiguration und Vorschau

Konfigurationsdialog und Vorschau

Abbildung 18 zeigt die überschaubare Konfiguration der Klasse und die dazugehörige Vorschau. Abschnitt C.2 enthält ausführliche Informationen über die Konfigurationsparameter.

4.1.3 *Geschwindigkeitsregulierung (Pfeil)*

Die letzte Anzeige in diesem Abschnitt ist in Abbildung 19 zu sehen. Sie stellt das Konzept von Team 35 dar und wird in der Klasse `AAFSpeedRegulationArrowsAgent` implementiert. Konzeptuell betrachtet ähnelt sie der Anzeige von Team 34 (Abbildung 17). Der Unterschied zwischen den beiden Konzepten besteht darin, dass Team 35 nur die Abweichung der vertikalen Joystick-Ist-Geschwindigkeit von der (und hier liegt der zweite Unterschied) für den aktuellen Streckenabschnitt empfohlenen vertikalen Geschwindigkeit berücksichtigt und anzeigt. Die Anzeige besteht auf beiden Seiten aus einem schwarzen Pfeil, der je nachdem ob die MWB langsamer oder schneller im Vergleich zu der für den Streckenabschnitt optimale Geschwindigkeit fahren, nach oben oder nach unten gerichtet ist. Der Hintergrund der Anzeige, der Bereich von dem unteren sichtbaren Bereich der Strecke bis zur Höhe der Pfeile wird in weißer durchsichtiger Farbe angezeigt. Für die Bestimmung der optimalen vertikalen Geschwindigkeit für den aktuellen Streckenabschnitt werden nicht wie bis jetzt die Methoden der Klasse `AAFOptimalStep` benutzt, sondern die Methode

- `calculateSpeedForY: y branch: bSymbol`

der Klasse `AAFTrackOptimalSpeed`. Die Klasse ist in der Kategorie *AAF-Agents-Concepts-Support* zu finden. Der erste Parameter der Me-

Abbildung 19: Team 35, Anzeige der vertikalen Geschwindigkeitskorrektur

thode gibt die vertikale Position auf der Strecke an, für die die optimale vertikale Geschwindigkeit, mit der das Objekt an dieser Stelle der Strecke fahren kann, bestimmt werden soll. Der zweite Parameter kann entweder die Zeichenkette `#left` oder `#right` als Wert haben. Bei dem ersten Wert wird die optimale Geschwindigkeit für den linken Zweig und bei dem zweiten Wert für den rechten Zweig einer Gabelung berechnet. Wenn keine Gabelung auf der Strecke ist, muss der zweite Parameter den Wert `#left` haben.

Konfiguration und Vorschau

Der Konfigurationsdialog von `AAFSpeedRegulationArrowsAgent`, zu sehen zusammen mit der entsprechenden Vorschau in Abbildung 20, beinhaltet einen Parameter, der bei den vorigen zwei Automatiken in diesem Abschnitt nicht vorhanden ist - das Setzen der Transparenz von der Anzeige oder von Teilen von ihr.

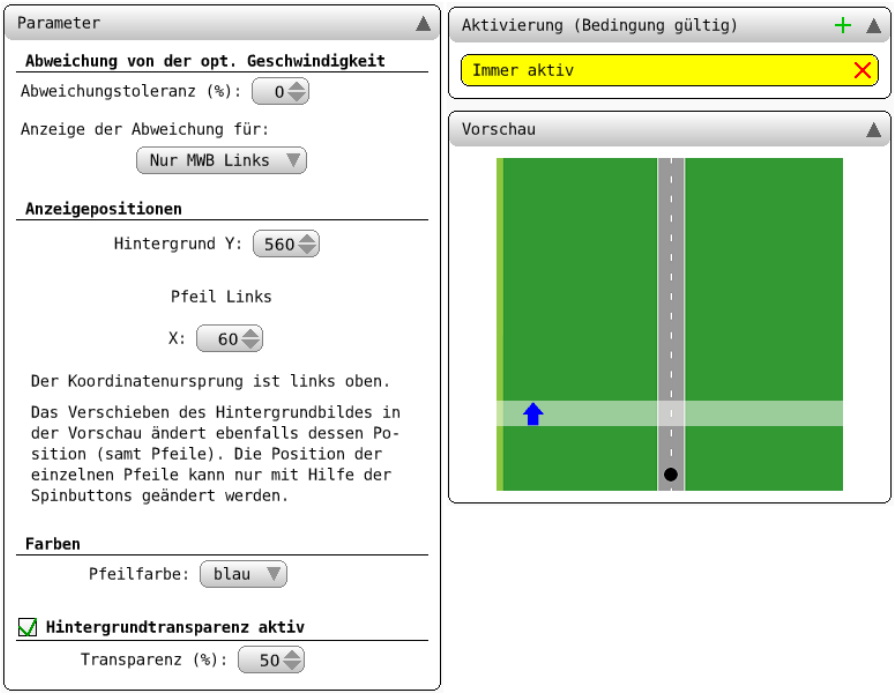

Abbildung 20: Geschwindigkeitsregulierung (Pfeil): Konfiguration und Vorschau

Wie in der Abbildung zu sehen, kann die Transparenz ein- und ausgeschaltet werden. Ist sie eingeschaltet, dann kann diese bis 100% er-

höht werden, was zum Verschwinden des Hintergrundbereichs führt. Die Transparenz selbst wird mit Hilfe der Methode

- `applyTransparency: aTransparency toForm: aForm`

von `AAFAgentUtils` aus der Kategorie *AAFGT-Widgets-Agents* gesetzt. Abschnitt C.3 beschreibt alle Parameter der Konfiguration im Detail.

4.2 GEMEINSAME ANZEIGE DER STEUERUNG(SVORGABEN)

In diesem Abschnitt wird die Implementierung der zweiten Klasse von Anzeigen aus Tabelle 2 vorgestellt. Im Unterschied zu den Konzepten im Abschnitt 4.1, in denen es für jeden der beiden MWB eine separate Anzeige gibt, sind die Anzeigen in diesem Abschnitt gemeinsam für die beiden MWB, d.h. die Information in der Anzeige ist für beide Versuchspersonen relevant und gleich. Die Anzeigenklasse besteht aus einem Konzept von Team 30 und aus einem Konzept von Team 38 (Tabelle 4). Diese werden in zwei Klassen implementiert.

Tabelle 4: Implementierte Klassen für gemeinsame Steuerungsanzeigen

KLASSE	TEAM
`AAFSpeedRegulationBoxesAgent` "Geschwindigkeitsregulierung (Kästchen)"	30
`AAFSteeringProposalDisplayAgent` "Steuerungsvorgabenanzeige"	38

4.2.1 *Geschwindigkeitsregulierung (Kästchen)*

Abbildung 21 beschreibt das Konzept von Team 30. Die Anzeige visualisiert die Abweichung der gemeinsamen vertikalen Geschwindigkeit der beiden MWB von der für den Streckenabschnitt optimalen vertikalen Geschwindigkeit. Sie enthält links und rechts jeweils vier unterschiedlich große, transparente Balken mit schwarzem Rand. Der Größe nach sind die Balken links absteigend und rechts aufsteigend geordnet. Ist die summierte vertikale Geschwindigkeit der beiden MWB gleich der optimalen Geschwindigkeit, sind alle Balken leer. Fahren dagegen beide Personen langsamer als von SAM empfohlen, wird auf der linken Seite der Anzeige ein Balken farblich hervorgehoben (Abbildung 22). Es ist ein Signal dafür, dass die MWB beschleunigen sollen. Umgekehrt, ein Balken auf der rechten Seite, der nicht leer ist, signalisiert, dass die beiden Personen zu schnell fahren und deshalb abbremsen sollen (Abbildung 23). In Abbildung 21 sind die Farben für die vier unterschiedlich großen Balken auf der linken Seite zu sehen. Die Balken auf der rechten Seite haben dieselben Farben. Die Balkengröße und damit auch die unterschiedliche Farbe für jede Bal-

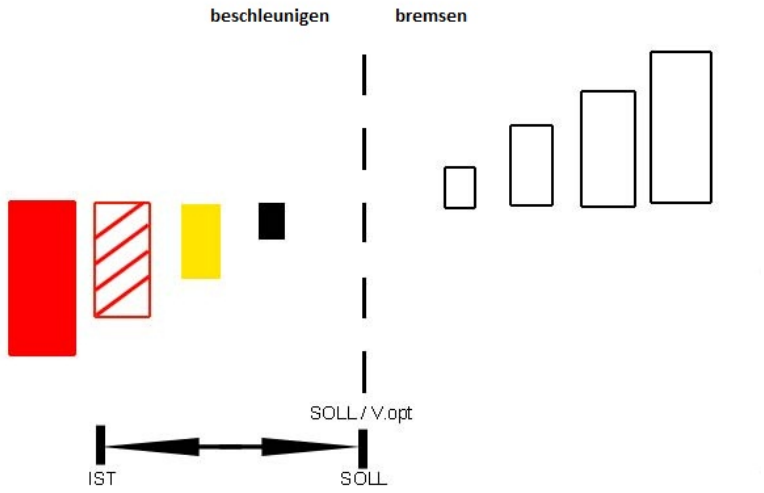

Abbildung 21: Team 30, Geschwindigkeitsregulierung (Kästchen) allgemein

kengröße gibt die Intensität der Handlung an. Ein größerer Balken bedeutet je nach Situation entweder mehr beschleunigen oder mehr bremsen.

Bei der Anzeige von Team 30 ist zu beachten, dass die Abweichung der Ist-Geschwindigkeit (die summierte vertikale Geschwindigkeit der beiden MWB) von der Soll-Geschwindigkeit (empfohlene vertikale Geschwindigkeit) nicht stetig, sondern diskret angezeigt wird. Die jeweils vier Balken für Beschleunigen und für Bremsen in der Anzeige werden entweder komplett gefärbt oder gar nicht. Eine stufenweise Färbung der Balken ist nicht möglich.

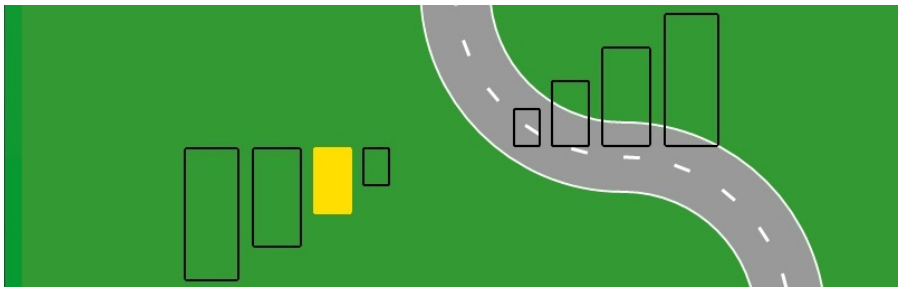

Abbildung 22: Team 30, Geschwindigkeitsregulierung: Beschleunigen

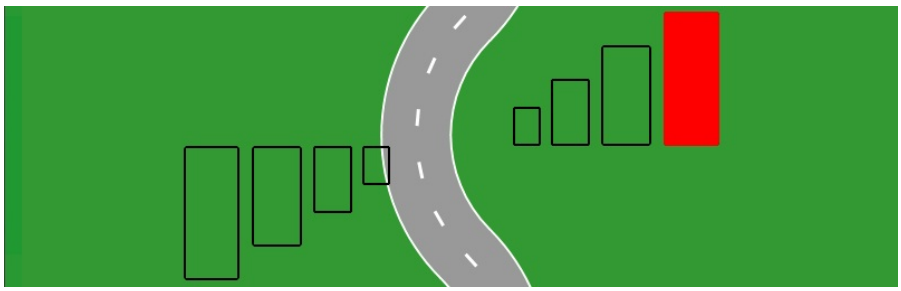

Abbildung 23: Team 30, Geschwindigkeitsregulierung: Bremsen

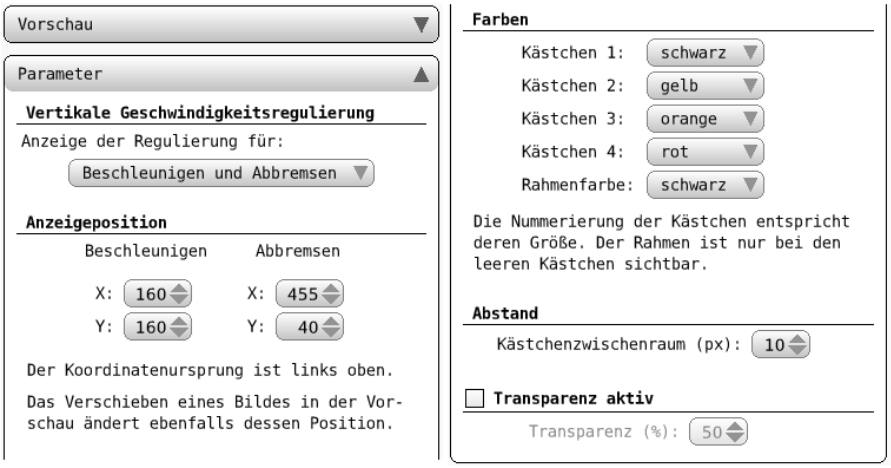

Abbildung 24: Geschwindigkeitsregulierung (Kästchen): Konfiguration

Konfiguration und Vorschau

Der Konfigurationsdialog der Klasse (Abbildung 24) bietet neben der individuellen Positionierung der beiden Teile der Anzeige, auch die Möglichkeit die Farben der gefüllten Kästchen sowie den Abstand zwischen diesen zu ändern. In Abschnitt C.4 sind alle Parameter des Dialogs beschrieben.

Die Vorschau entspricht dem Inhalt von Abbildung 21 und wird deshalb nicht noch einmal abgebildet.

4.2.2 *Steuerungsvorgabenanzeige*

In Abbildung 25 ist die Anzeige von Team 38 dargestellt, die in der Klasse `AAFSteeringProposalDisplayAgent` implementiert wird. Der horizontale rote Balken am unteren Bildrand zeigt die laterale Soll-Position des Joysticks, d.h. die beiden MWB sollen bei der seitlichen Steuerung versuchen, mit ihren Joystickbewegungen nach links und rechts der Anzeige zu folgen. Der vertikale rote Balken (rechts in der Abbildung) ist für das Anzeigen der maximalen vertikalen Geschwindigkeit, mit der die MWB auf dem aktuellen Streckenabschnitt fahren dürfen, vorgesehen.

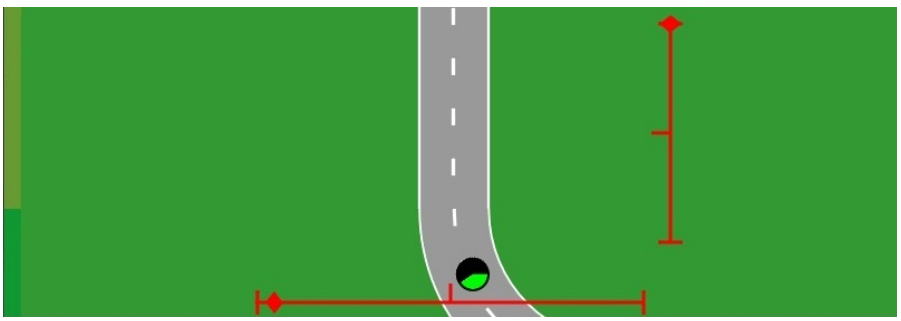

Abbildung 25: Team 38, Steuerungsvorgabenanzeige

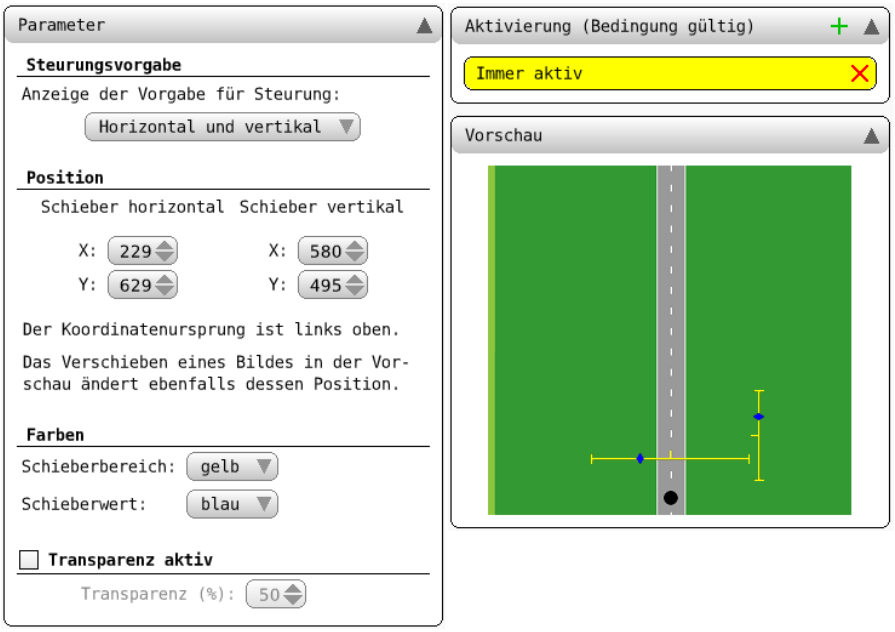

Abbildung 26: Steuerungsvorgabenanzeige: Konfiguration und Vorschau

Für die Berechnung der maximal zulässigen Geschwindigkeit wird die Steuergüte der beiden Fahrer benötigt. Diese gibt die Abweichung der beiden Fahrer von der Sollbahn an. Je größer diese Abweichung ist, desto kleiner ist die Steuergüte und somit auch die vertikale maximale Geschwindigkeit. Die Automatik für die Berechnung der Steuergüte ist noch nicht implementiert und wird nicht im Rahmen dieser Arbeit behandelt. Aus diesem Grund kann die maximal zulässige vertikale Geschwindigkeit nicht ermittelt werden und auf dem vertikalen Balken wird nur die optimale vertikale Geschwindigkeit ohne Berücksichtigung der Steuergüte angezeigt. Für die Ermittlung sowohl der optimalen seitlichen Steuerung als auch der optimalen vertikalen Geschwindigkeit werden die im Abschnitt 4.1 vorgestellten Methoden

- `optimalFromX: x Y: y`
- `optimalForkFromX: x Y: y branch: b`

verwendet.

Konfiguration und Vorschau

Die Vorschau der Automatik und die Konfigurationsparameter dazu sind in Abbildung 26 zu sehen. Die Beschreibung aller Parameter des Dialogs und deren Werte befindet sich im Abschnitt C.5.

4.3 ANZEIGE AM/IM TRACKINGOBJEKT

Die Konzepte, die eine visuelle Anzeige am/im Trackingobjekt vorsehen, bilden die dritte Klasse von Anzeigen aus Tabelle 2 und werden in diesem Abschnitt näher erläutert. Tabelle 5 listet alle Squeak-Klassen auf, die die Konzepte für diesen Abschnitt implementieren.

Tabelle 5: Implementierte Klassen für Anzeigen am/im Trackingobjekt

KLASSE	TEAM
`AAFArrowsOnObjectAgent` "Pfeile am Objekt"	20, 23, 26
`AAFSpeedAndControlRegulationCapsAgent` "Steuerungsregulierung am Objekt"	24
`AAFTempomatDisplayAgent` "Tempomat-Anzeige"	32
`AAFTachometerDisplayAgent` "Tachometer-Anzeige"	28
`AAFColoredObjectAgent` "Farbänderung Objekt"	24
`AAFOptimalPosInObjectAgent` "Joystickanzeige im Objekt"	39

4.3.1 *Pfeile am Objekt*

Die Anzeigen in Abbildung 27, in Abbildung 28 und in Abbildung 29 werden in der Klasse `AAFArrowsOnObjectAgent` implementiert. Das Konzept von Team 20 sieht einen schwarzen Pfeil als Anzeige vor, der die optimale Route zeigt. Die Anzeige von Team 23 enthält dieselbe Information und sie zeigt außerdem noch die optimale Geschwindigkeit (Sollgeschwindigkeit) an, mit der die Versuchspersonen die Strecke fahren sollen. Die Geschwindigkeitsanzeige wird durch das Verändern der Pfeillänge, was in Abbildung 28 nicht zu erkennen ist, erreicht. Beim Konzept von Team 26 sind vier Pfeile am Objekt zu beobachten. Die beiden schwarzen Pfeile zeigen die Richtung und die Geschwindigkeit, mit denen jeder der beiden MWB das Objekt steuert.

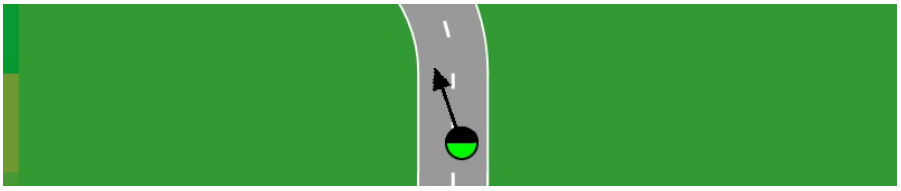

Abbildung 27: Team 20, Pfeil am Objekt: optimale Route

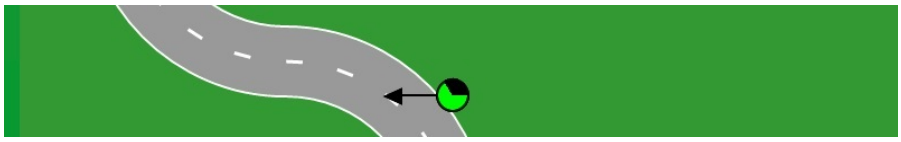

Abbildung 28: Team 23, Pfeil am Objekt: optimale Route und Geschwindigkeit

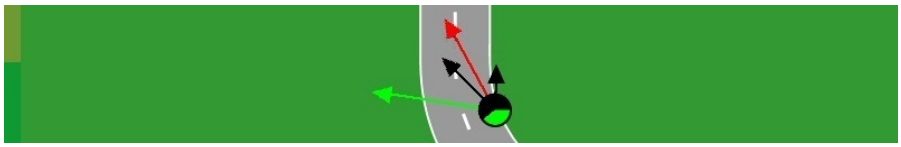

Abbildung 29: Team 26, Pfeil am Objekt: vier Pfeile

Diese zeigen sozusagen die Joystickbewegungen der beiden MWB an. Der rote Pfeil zeigt die Summe der beiden schwarzen Pfeile, während der grüne Pfeil die optimale Richtung und die optimale Geschwindigkeit darstellt. Die Geschwindigkeit wird in jedem der vier Pfeile durch ihre Pfeillänge visualisiert. Steuern beide MWB optimal, dann ist sowohl die Pfeillänge als auch die Pfeilrichtung des roten und des grünen Pfeils identisch. In diesem Fall ist nur einer der beiden Pfeile in der Anzeige zu sehen und zwar der Pfeil, der als letzter von den beiden gezeichnet wurde. In der aktuellen Implementierung wird der grüne Pfeil als letzter in die Anzeige eingefügt und somit, im Falle einer Übereinstimmung der beiden Pfeile, ist er der zu sehende Pfeil. Diese Entscheidung wurde vom Implementierer getroffen, da es im Konzept hierfür keine Reihenfolge festgelegt worden ist. In Listing 9 ist die `compute:`-Methode der Klasse zu sehen.

```
1  compute: aSamState
2    | arrowsDrawFormCopy |
3
4    arrowsDrawFormCopy := (arrowsDrowForm copy:(arrowsDrowForm
         boundingBox)).
5
6    self drawMwiLeftPos: (aSamState joystickRaw1)
7         andMwiRightPos: (aSamState joystickRaw2)
8         on: arrowsDrawFormCopy.
9
10   self drawMwiOptimalPosOn: arrowsDrawFormCopy
11        forState: aSamState.
12
13   arrowsDisplay
14        sourceForm: arrowsDrawFormCopy;
15        destX: ((AAFValues getTrackingObjXPosForCurrentState:
              aSamState) - (arrowsDrowForm width / 2)).
16
17   aSamState bitBlts add: arrowsDisplay.
18   ^ aSamState.
```

Listing 9: AAFArrowsOnObjectAgent: Methode `compute:`

Am Anfang der Methode wird die Zeichenfläche, ein `Form`-Objekt, für die Pfeile erzeugt (Zeile 4). Danach werden die Methoden der Klasse zum Zeichnen der Pfeile aufgerufen (Zeile 6 und 10), die die Joystickpositionen der beiden MWB, deren Summe und die optimale Objektrichtung und -geschwindigkeit visualisieren. Diese Methoden zeichnen nur diejenigen Pfeile, die der Anwender bei der Konfiguration der Automatik aktiviert hat. Nachdem die Pfeile gezeichnet wurden, wird die Zeichenfläche dem `BitBlt`-Objekt `arrowsDisplay` hinzugefügt (Zeile 13). Zusätzlich wird diesem auch die aktuelle Position des Trackingobjekts übergeben, um die Anzeige genau positionieren zu können. Zum Schluss wird `arrowsDisplay` in die `bitBlts`-Variable von `SamState` gespeichert, um später in SAM ausgelesen und angezeigt werden zu können[4].

Konfiguration und Vorschau

Der Konfigurationsdialog der Klasse und ihre Vorschau sind in Abbildung 30 abgebildet.

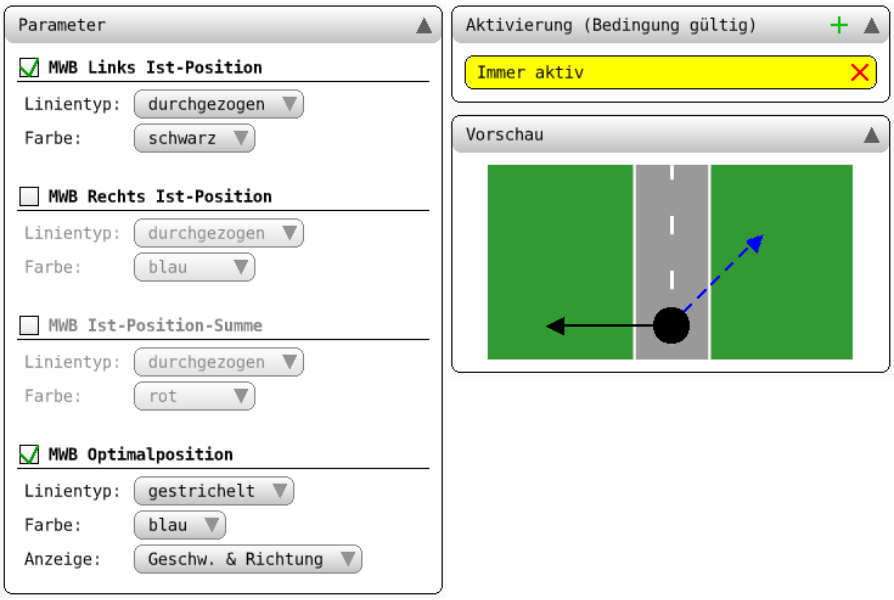

Abbildung 30: Pfeile am Objekt: Konfiguration und Vorschau

Bei der Konfiguration der Klasse kann in erster Linie nur ausgewählt werden, welche Pfeile sollen angezeigt oder nicht angezeigt werden sollen. Soll ein Pfeil in der Anzeige zu sehen sein, dann kann sein Aussehen zusätzlich konfiguriert werden. Die Parameter hierfür sind im Abschnitt C.6 beschrieben.

4 Abschnitt 2.3.2 beschreibt den Prozess ausführlicher.

Abbildung 31: Team 24, Steuerungsregulierung am Objekt

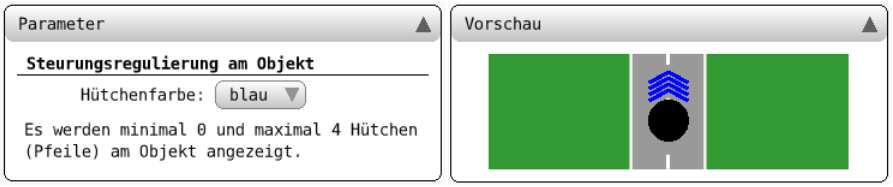

Abbildung 32: Steuerungsregulierung am Objekt: Konfiguration und Vorschau

4.3.2 *Steuerungsregulierung am Objekt*

Die Klasse `AAFSpeedAndControlRegulationCapsAgent` implementiert die Anzeige des nächsten Konzepts in diesem Abschnitt. Zu sehen in Abbildung 31, zeigt die Automatik von Team 24 die optimale Route und Geschwindigkeit für das Objekt an. Im Vergleich zu der Klasse aus dem vorigen Abschnitt, die ebenfalls die optimale Objektrichtung und -geschwindigkeit visualisiert, wird bei Team 24 nicht die tatsächliche optimale Geschwindigkeit angezeigt, sondern die Differenz von ihr und von der aktuellen Geschwindigkeit des Objekts. Wenn die Objektgeschwindigkeit gleich der optimalen Geschwindigkeit ist, dann sehen die MWB keine Anzeige. Bei Abweichung werden bis zu vier Pfeile/Hütchen angezeigt. Die Anzahl der Pfeile gibt die Größe der Abweichung wieder.

Konfiguration und Vorschau

Der Konfigurationsdialog der Klasse sowie ihre Vorschau sind in Abbildung 32 zu betrachten. Obwohl dieser nur das Konfigurieren der Pfeilfarbe erlaubt, wird ihr Parameter der Vollständigkeit halber im Abschnitt C.7 beschrieben.

4.3.3 *Tempomat-Anzeige*

Das Konzept von Team 32, die letzte Pfeilanzeige dieses Abschnittes, wird in Abbildung 33 dargestellt. Die Anzeige besteht aus einem Pfeil, der über dem Objekt angezeigt wird. Der Pfeil zeigt die Richtung in der sich das Objekt bewegen soll. Diese kann geradeaus, nach links oder nach rechts sein. Er zeigt zusätzlich ob die Objektgeschwindigkeit erhöht oder verringert werden soll. Dafür wird der jeweilige Pfeil in grün für Beschleunigen oder in rot für Bremsen gefärbt. Die Richtung und die Farbe des angezeigten Pfeils wird durch

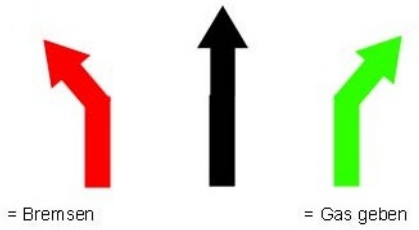

Abbildung 33: Team 32, Tempomat-Anzeige allgemein

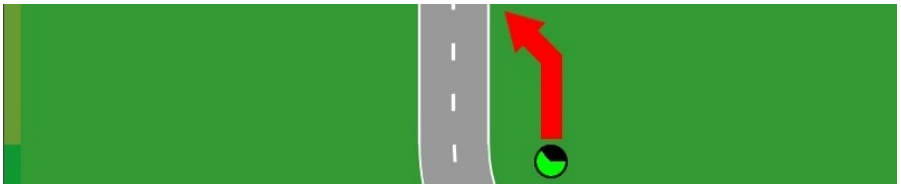

Abbildung 34: Tempomat-Anzeige: Bremsen und nach links

den Vergleich der aktuellen Objektposition und -geschwindigkeit mit der Soll-Position und -geschwindigkeit für das Objekt bestimmt. In Abbildung 34 befindet sich das Objekt rechts außerhalb der Strecke. Damit es möglichst schnell auf die Strecke zurückkommt, muss sich das Objekt nach links zur Strecke bewegen (Pfeil nach links) und es muss dabei seine Geschwindigkeit reduzieren (Pfeil rot). In Listing 10 ist die Methode zu sehen, die zum Färben der Pfeile implementiert wurde.

```
1  colorArrowForm: arrowForm forDiff: yDiff
2    | tmpForm |
3
4    (yDiff between: (self class defaultToleranceLimit negated) and:
         (self class defaultToleranceLimit))
5      ifTrue: [
6        tmpForm := arrowForm.
7      ]
8      ifFalse: [
9        (yDiff > 0)
10         ifTrue: [
11           tmpForm := ((arrowForm copy: (arrowForm boundingBox))
                 replaceColor: (Color black) withColor:
                 brakeArrowColor).
12         ]
13         ifFalse: [
14           tmpForm := ((arrowForm copy: (arrowForm boundingBox))
                 replaceColor: (Color black) withColor:
                 speedupArrowColor).
15         ].
16     ].
17
18   ^ tmpForm.
```

Listing 10: Tempomat-Azeige: Methode zum Färben der Pfeile

Die Methode in Listing 10 wird in der `compute:`-Methode der Klasse aufgerufen. Ihr wird dabei ein Pfeil in Form eines `Form`-Objekts und die Differenz zwischen der optimalen und aktuellen Objektgeschwindigkeit übergeben. In Zeile 4 wird zuerst überprüft, ob die übergebene Differenz innerhalb eines vom Entwickler bestimmten Toleranzbereichs liegt. Wenn das der Fall ist, dann wird der übergebene Pfeil einfach wieder zurückgeliefert. Im anderen Fall wird der Pfeil je nachdem, ob die Differenz positiv oder negativ ist, in der Farbe fürs Bremsen (Zeile 11) oder für die Beschleunigung (Zeile 14) gefärbt. Für diesen Zweck wird die Methode

- `replaceColor: aColor withColor: aColor`

der Squeak-Klasse `Form` verwendet. Wie in den Zeilen 11 und 14 zu sehen ist, wird die gerade erwähnte Methode nicht am Pfeilobjekt selbst aufgerufen, sondern an einer Kopie von ihm. Dadurch muss sich das Programm die letzte Farbe des jeweiligen Pfeils nicht merken und kann bei jedem Durchlauf einfach die Originalfarbe des Pfeils (in diesem Fall schwarz) als die zu ersetzende angeben.

Konfiguration und Vorschau

Die Parameterbeschreibung der einfachen Konfiguration der Klasse (Abbildung 35) ist im Abschnitt C.8, Anhang C nachzulesen.

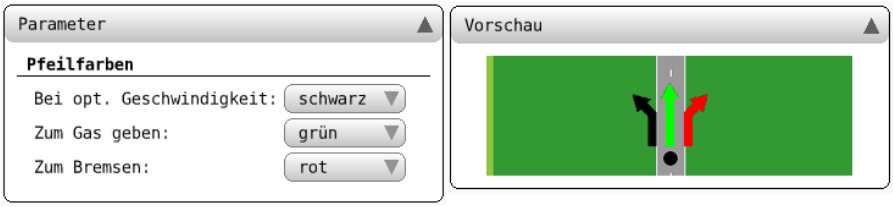

Abbildung 35: Tempomat-Anzeige: Konfiguration und Vorschau

4.3.4 *Tachometer-Anzeige*

Die letzte Anzeige in diesem Abschnitt, die am und nicht im Trackingobjekt gezeigt wird, ist in Abbildung 36 zu sehen. Sie stellt einen Tachometer dar, der sich unter dem Trackingobjekt befindet und sich mit ihm nach links und rechts bewegt. Der schwarze Balken zeigt die Ist-Geschwindigkeit des Objekts. Bei maximaler Objektgeschwindigkeit ist der Balken auf 0, beim Stillstand auf 180 im mathematisch positiven Drehsinn. In der Abbildung von Team 28 für diese Anzeige kann man sehen, dass der Tachometer in drei unterschiedlichen Farben gefärbt sein kann (in unserem Beispiel grün, orange und rot). Die unterschiedliche Farbfüllung des Tachometers basiert auf der Abweichung der Objektgeschwindigkeit von der für den Streckenabschnitt optimalen Geschwindigkeit. Da es im Konzept nicht definiert ist, bei welcher Abweichung welche der Farben anzuzeigen ist, wurde diese

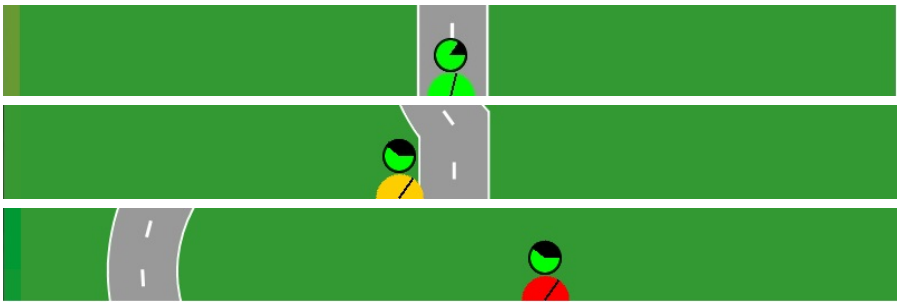

Abbildung 36: Team 28, Tachometer-Anzeige

Entscheidung dem Anwender überlassen. Dieser kann im Konfigurationsdialog der Klasse (Abbildung 37) zwei Grenzwerte für die Abweichung setzen. Der erste Wert gibt an, bis zu welcher Abweichung der Tachometer in der ersten Farbe, die für eine optimale Geschwindigkeit steht, angezeigt wird. Ist die Abweichung größer als der erste Wert, dann wird die Anzeige mit der zweiten Farbe gefärbt, bis die Abweichung auch den zweiten Wert überschritten hat. In diesem Fall erscheint der Tachometer in der dritten und letzten Farbe, die in der Konfiguration ausgewählt wurde. Die Methode in Listing 11 implementiert die soeben beschriebene Färbung des Tachometers, oder besser gesagt die Auswahl zwischen den bereits gefärbten Tachometern.

```
1  getTachometerFormForTrackSpeed: trackYSpeed objSpeed: objYSpeed
2    (objYSpeed between: ((1 - toleranceLimit1) * trackYSpeed) and:
         (((1 + toleranceLimit1) * trackYSpeed)))
3      ifTrue: [
4        ^ tachometerOptForm.
5      ].
6
7    (objYSpeed between: ((1 - toleranceLimit2) * trackYSpeed) and:
         (((1 + toleranceLimit2) * trackYSpeed)))
8      ifTrue: [
9        ^ tachometerWarning1Form.
10     ].
11
12   ^ tachometerWarning2Form.
```

Listing 11: Tachometer-Anzeige: Methode zur Bestimmung der Frabe des Tachometers

Konfiguration und Vorschau

Die Konfiguration der Klasse, die zum Teil bereits in diesem Unterabschnitt vorgestellt wurde, und die dazugehörige Vorschau sind in Abbildung 37 zu sehen. Im Abschnitt C.9 ist eine Beschreibung aller Konfigurationsparameter zu finden.

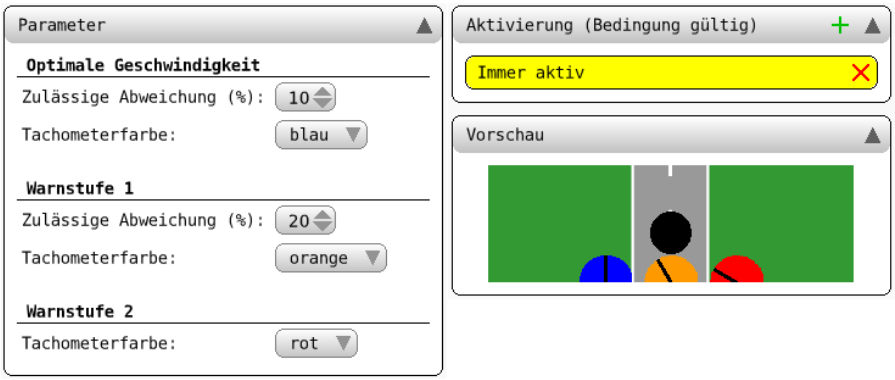

Abbildung 37: Tachometer-Anzeige: Konfiguration und Vorschau

4.3.5 *Farbänderung Objekt*

Die Anzeigen der bisher vorgestellten Konzepte in diesem Abschnitt waren alle am Objekt positioniert. In diesem Unterabschnitt wird das erste Konzept vorgestellt, das eine Anzeige *im* Objekt selbst vorsieht. Die Idee des Konzepts von Team 24 ist in Abbildung 38 zu sehen. Bei dieser Automatik wird das Objekt in Abhängigkeit der Anzahl seiner Sensoren, die außerhalb des Streckenbereichs liegen, in drei unterschiedlichen Farben gefärbt. Die soeben angesprochenen Sensoren sind 8 an der Zahl und sind in gleichen Abständen von einander entfernt am Objektrand positioniert. Wenn einer dieser Sensoren die grüne Farbe sieht, d.h. er sich links oder rechts von der weißen Linie befindet, die den grauen Streckenbereich eingrenzt, dann wird dies in jedem Tick von SAM in die Variable `noSensorsOffTrack` von `SAMModelData` gespeichert. Genau diese Variable wird in der `compute:`-Methode der Klasse (Listing 12) benutzt, um die Farbe des Trackingobjekts zu bestimmen. Ist der Wert der Variable kleiner gleich dem ersten Grenzwert (Zeile 7), der vom Anwender konfiguriert wurde, dann wird das Objekt in der ersten Farbe gefärbt. Wenn die Anzahl der Sensoren außerhalb der Strecke größer ist als der erste Grenzwert, aber kleiner gleich dem zweiten Grenzwert (Zeile 15), der auch vom Anwender eingegeben wurde, dann bekommt das Objekt die zweite Farbe. Bei Überschreitung des zweiten Grenzwertes wird das Objekt schließlich in der letzten Farbe gefärbt.

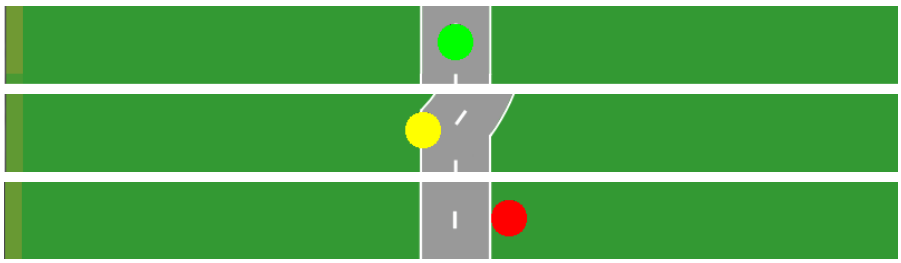

Abbildung 38: Team 24, Farbänderung Objekt

Es sei an dieser Stelle erwähnt, dass das Objekt selbst nicht gefärbt wird. Es wird nur von einem Kreis mit der gewünschten Farbe überdeckt, der dieselbe Größe hat wie das Trackingobjekt. Damit dieser wirklich über das Objekt gezeigt werden kann, muss das `BitBlt`-Objekt, welches die Anzeige enthält, nicht wie bei den Automatiken bis jetzt in die `bitBlts`-Varible von `SamState`, sondern in `postObjBlts` gespeichert werden (Zeile 11, 19, 25) .

```
1  compute: aSamState
2    | newX |
3
4    "the new x pos of the object"
5    newX:=(AAFValues getTrackingObjXPosForCurrentState: aSamState).
6
7    ((modelData noSensorsOffTrack) <= range1Sensors)
8      ifTrue: [
9        coloredDisplay sourceForm: range1Form;
10                      destX: (newX - (range1Form width / 2)).
11       aSamState postObjBlts add: coloredDisplay.
12       ^ aSamState.
13     ].
14
15   ((modelData noSensorsOffTrack) <= range2Sensors)
16     ifTrue: [
17       coloredDisplay sourceForm: range2Form;
18                      destX: (newX - (range2Form width / 2)).
19       aSamState postObjBlts add: coloredDisplay.
20       ^ aSamState.
21     ].
22
23   coloredDisplay sourceForm: range3Form;
24                  destX: (newX - (range3Form width / 2)).
25
26   aSamState postObjBlts add: coloredDisplay.
27   ^ aSamState.
```

Listing 12: AAFColoredObjectAgent: Methode `compute:`

Konfiguration und Vorschau

Abbildung 39 stellt den Konfigurationsdialog und die Vorschau der Klasse dar. Erhöht man den Wert eines der Parameter "Sensoren außerhalb der Strecke", dann wandert das entsprechende Kreisobjekt in der Vorschau nach rechts bis es schließlich ganz außerhalb der Strecke steht. Der Teil des Kreises, der auf dem grünen Bereich liegt, enthält nicht immer die Anzahl der Sensoren die als Konfigurationswert angegeben wurde und ist nur als Orientierung für den Anwender gedacht. Im Abschnitt C.10 wird die komplette Liste der Konfigurationsparameter für die Klasse beschrieben.

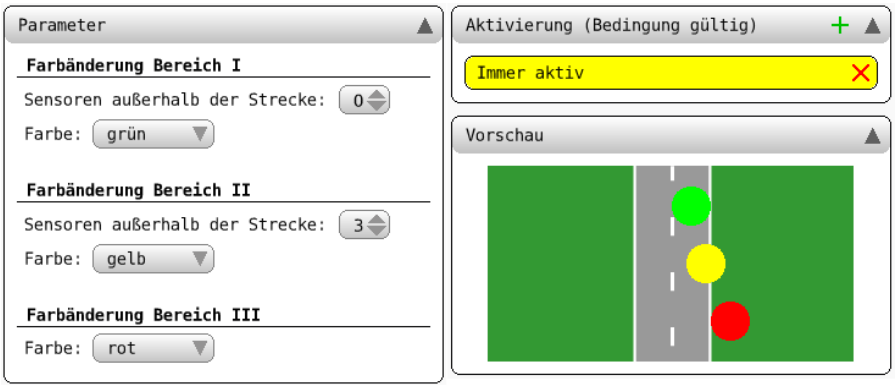

Abbildung 39: Farbänderung Objekt: Konfiguration und Vorschau

4.3.6 *Joystickanzeige im Objekt*

Abbildung 40 zeigt die Anzeige des letzten Konzepts in diesem Abschnitt. Es ist die zweite Automatik mit einer Anzeige im Objekt. Die Implementierung findet in der Klasse `AAFOptimalPosInObjectAgent` statt. Die Idee dieser Anzeige ist das Anzeigen der idealen Joystickposition im Objekt selbst. Das zu steuernde Objekt hat einen Durchmesser von 30 Pixeln. Nimmt man dieses runde Objekt als Koordinatensystem, dann hat man für die Darstellung der idealen Joystickposition in jedem Quadrant maximal 15 Pixel. Das Objekt, welches die ideale Postion des Joysticks anzeigen soll, muss deshalb viel kleiner als 15 Pixel sein.

Abbildung 40: Team 39, Joystickanzeige im Objekt

Konfiguration und Vorschau

Die Überlappung der Anzeige der Objekt-Ist-Geschwindigkeit mit der Anzeige der optimalen Joystickposition im Objekt selbst, kann bei einer ungünstigen Wahl der Farbe für die optimale Position dazu führen, dass die beiden Anzeigen schwer oder gar nicht auseinandergehalten werden können. Um dieses Problem zu lösen, bietet der Konfigurationsdialog der Klasse (Abbildung 41) die Möglichkeit, die Objektgeschwindigkeit zu verstecken und das Objekt selbst, ähnlich der Automatik aus dem vorigen Unterabschnitt, in einer bestimmten Farbe zu färben. Eine genaue Beschreibung der Konfigurationsparameter kann im Abschnitt C.11 nachgelesen werden.

Abbildung 41: Joystickanzeige im Objekt: Konfiguration und Vorschau

4.4 VISUELLE HINWEISE ZUR VERMEIDUNG VON KOLLISIONEN MIT DYNAMISCHEN HINDERNISSEN

In diesem Abschnitt werden die Anzeigen behandelt, die für die Vermeidung von Kollisionen mit dynamischen Hindernissen konzipiert wurden. Diese Anzeigen bilden die vierte Klasse von Anzeigen aus Tabelle 2. Sie alle basieren konzeptuell auf ein und demselben Algorithmus und werden deshalb in einer einzigen Klasse (Tabelle 6) implementiert.

Tabelle 6: Implementierte Klasse für Automatiken mit visuellen Hinweisen zur Vermeidung von Kollisionen mit dynamischen Hindernissen

Klasse	Team
`AAFDynObstCollisionWarningAgent` "Hindernis-Kollisionswarnhinweis (dynamisch)"	22, 38, 41, 42, 45, 46, 49

Die Implementierung des gemeinsamen Algorithmus[5] in Squeak findet in der Klasse `AAFSpeedHintsAtDynamicObstacles` aus der Kategorie *AAF-Agents-Concepts-Support* statt. Um die Berechnung des Algorithmus benutzen zu können, wird die Klassenmethode

- `ySpeedRecommendationForState: aSAMState`

aufgerufen. Die Methode erwartet den aktuellen Zustand von SAM `aSamState`. Es wird berechnet, ob bei Beibehaltung der aktuellen Objektgeschwindigkeit eine Kollision mit dem nächsten dynamischen Hindernis möglich ist. Wenn keine Kollision vorliegt, dann wird die Zeichenkette `#Keep` zurückgegeben, andernfalls liefert die Methode entweder `#Brake` oder `#SpeedUp` zurück. `#Break` bedeutet, dass die Objektgeschwindigkeit verringert werden soll, um eine mögliche Kollision auszuweichen, wahrend `#SpeedUp` für eine Erhöhung der Geschwindigkeit steht. Die Berechnung der Kollisionsmöglichkeit findet immer nur dann statt, wenn sich ein dynamisches Hindernis im für die MWB sichtbaren Streckenbereich befindet.

5 Die 1. Version des Algorithmus wurde vom Autor implementiert. Diese wurde später im Rahmen seiner Diplomarbeit von Kosjar (2012) überarbeitet und erweitert.

```
1  setCollisionWarning: aSamState
2    | speedRecommendation |
3
4    speedRecommendation := AAFSpeedHintsAtDynamicObstacles
         ySpeedRecommendationForState: aSamState.
5
6    (speedRecommendation = #SpeedUp) & (collisionWarningType
         includesSubstring: speedRecommendation caseSensitive:false)
7      ifTrue: [
8        aSamState bitBlts add: speedup.
9        logger setLogEntry: (logger class ccDynObstacleCAFaster)
             file: (self speedupHintImageFile).
10     ].
11
12   (speedRecommendation = #Brake) & (collisionWarningType
         includesSubstring: speedRecommendation caseSensitive:false)
13     ifTrue: [
14       aSamState bitBlts add: brake.
15       logger setLogEntry: (logger class ccDynObstacleCASlower)
             file: (self brakeHintImageFile).
16     ].
```

Listing 13: AAFDynObstCollisionWarningAgent: Methode zum Anzeigen von Hinweisen zur Kollisionsvermeidung

Die Methode zum Anzeigen von Hinweisen zur Kollisionsvermeidung ist in Listing 13 angegeben. Diese zeigt nur dann etwas an, wenn ein Hinweis zur Änderung der Geschwindigkeit vorliegt und die Art des Hinweises bei der Konfiguration der Automatik erlaubt wurde (Zeile 6 und 12). Wenn die Automatik etwas anzeigt, dann wird die Art des Hinweises und der Name des anzuzeigenden Bildes in die Log-Datei von SAM gespeichert (Zeile 9 und 15). Das ist die einzige Automatik in dieser Arbeit, deren Aktivität geloggt wird. Es liegt daran, dass sie im Unterschied zu den anderen Automatiken, die während einer ganzen Fahrt aktiv sind, nur beim Auftreten eines dynamischen Hindernis aktiviert wird. Eine Beschreibung der Log-Datei von SAM und der Protokollierung von Automatikdaten in diese ist in der Studienarbeit von Seid (2012) zu finden.

In den Anzeigen von Team 22, Team 42 und Team 46 wird mit Textnachrichten auf die Geschwindigkeitsänderung hingewiesen. Diese sind in den Abbildungen 42, 43 und 44 zu sehen. In den Anzeigen von Team 22 und von Team 42 wird in der rechten oberen Bildschirmecke auf Beschleunigen und in der rechten unteren Bildschirmecke auf Bremsen hingewiesen. In der Anzeige von Team 46 (Abbildung 44) werden beide Hinweise zur Änderung der Objektgeschwindigkeit an derselben Stelle gezeigt und zwar horizontal mittig auf der Strecke und vertikal etwas über der Position des Trackingobjekts.

Abbildung 42: Team 22, Hinweise zur Kollisionsvermeidung

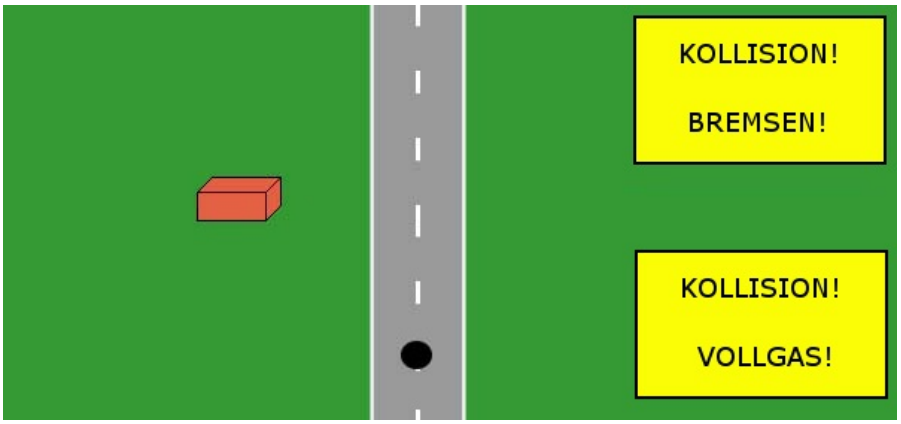

Abbildung 43: Team 42, Hinweise zur Kollisionsvermeidung

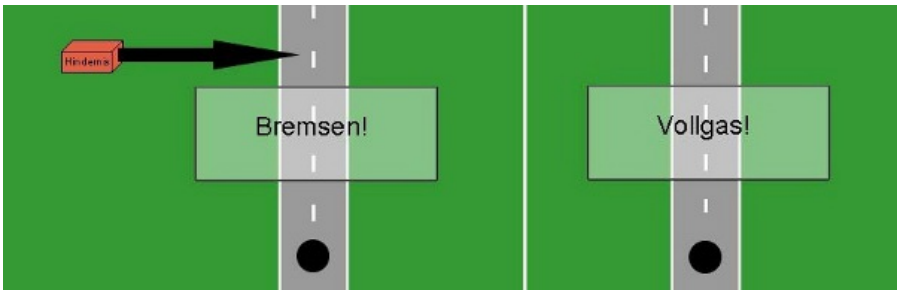

Abbildung 44: Team 46, Hinweise zur Kollisionsvermeidung

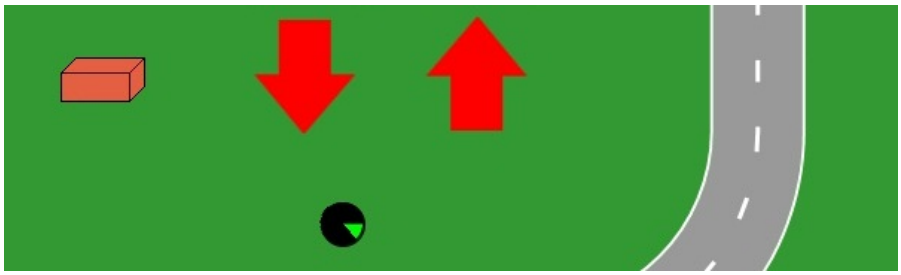

Abbildung 45: Team 38, Hinweise zur Kollisionsvermeidung

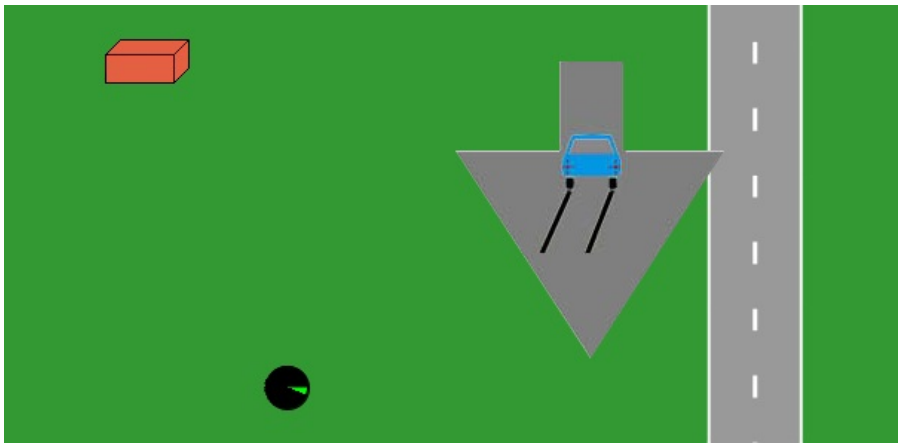

Abbildung 46: Team 41, Hinweise zur Kollisionsvermeidung

Alle weiteren Anzeigen in diesem Unterabschnitt benutzen Pfeile, um die Versuchspersonen auf eine Änderung ihrer vertikalen Geschwindigkeit (und damit auch der Objektgeschwindigkeit) hinzuweisen. In den Konzepten, die in den Abbildungen 45, 46 und 47 zu sehen sind, wird auf eine Erhöhung der Objektgeschwindigkeit mit einem Pfeil nach oben und auf eine Verringerung der Geschwindigkeit mit einem Pfeil nach unten hingewiesen. Die Pfeile von Team 38 sind rot, von Team 41 grau und von Team 45 schwarz.

In Abbildung 48 ist die letzte Anzeige dieses Abschnitts dargestellt. Sowohl der Beschleunigungshinweis als auch der Bremshinweis werden in der Anzeige durch einen roten Pfeil angezeigt, der in beiden Fällen nach oben gerichtet ist. Die Unterscheidung zwischen den bei-

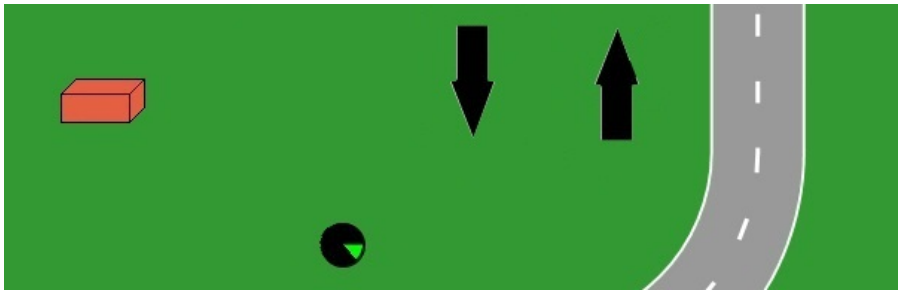

Abbildung 47: Team 45, Hinweise zur Kollisionsvermeidung

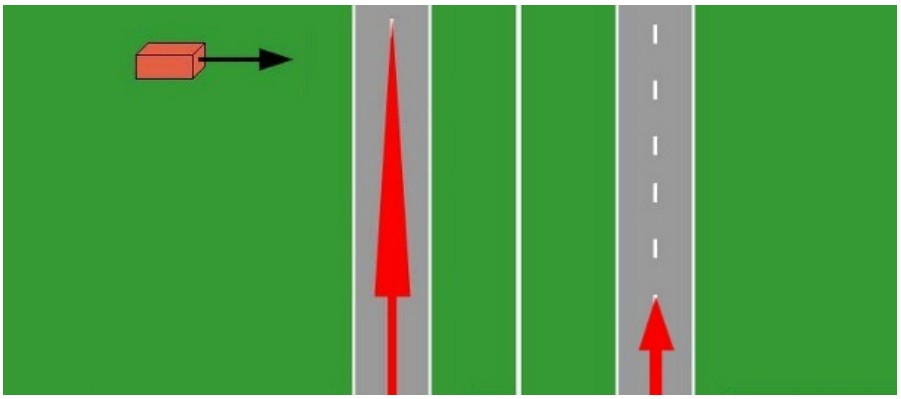

Abbildung 48: Team 49, Hinweise zur Kollisionsvermeidung

den Geschwindigkeitshinweisen geschieht demnach nicht nach der Ausrichtung und der Farbe des Pfeils, sondern nach seiner Länge und seiner Form. Ein langer und spitzer Pfeil signalisiert Beschleunigen, wogegen ein kurzer und flacher Pfeil Bremsen bedeutet.

Konfiguration und Vorschau

Abbildung 49 zeigt den Konfigurationsdialog der Klasse und die dazu entsprechende Vorschau. Der große Unterschied zu den bisher vorgestellten Dialogfenstern ist der Bereich mit der Aktivierungsbedingung. Während es bis jetzt in diesem Bereich der Text "Immer aktiv" zu lesen war, steht bei diesem Dialog auf einmal der Text "Bevor/im Hindernisbereich". Außerdem bietet dieser neue Ereignistyp mehrere Auswahlmöglichkeiten, um die Art des Hindernisses zu bestimmen, bei dessen/deren Auftreten die Automatik aktiviert werden muss. Da die Automatik speziell für dynamische Hindernisse implementiert wurde, ist logischerweise die Möglichkeit "Dynamisch" ausgewählt worden.

```
1  setEvent
2    | ev |
3
4    ev := AAFTrackObstacleAhead new.
5    ev lowerBoundOffset: -715;
6       upperBoundOffset: AAFTrackObstacleAhead obstacleHeight.
7
8    "remove all default types of interest except #
         nextDynamicObstacle."
9    ev removeTypeOfInterest: #nextStaticObstacle25CoverageLeft.
10   ev removeTypeOfInterest: #nextStaticObstacle50CoverageLeft.
11   ev removeTypeOfInterest: #nextStaticObstacle25CoverageRight.
12   ev removeTypeOfInterest: #nextStaticObstacle50CoverageRight.
13
14   self event: ev.
```

Listing 14: Initialisierung des Aktivierungsereignisses

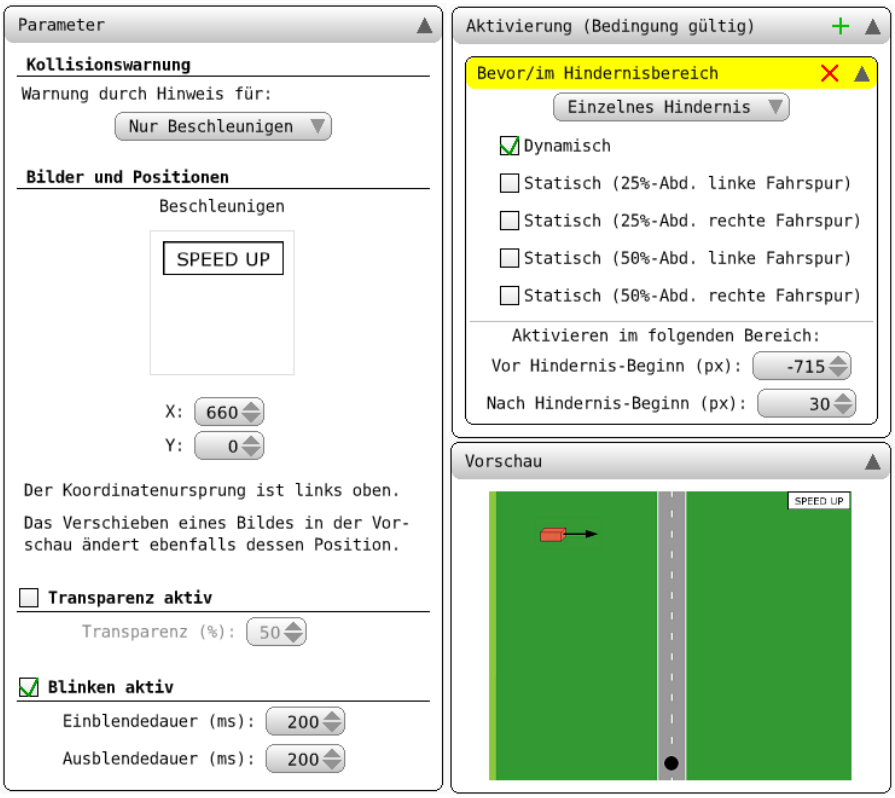

Abbildung 49: Hindernis-Kollisionswarnhinweis (dynamisch): Konfiguration und Vorschau

Damit der Anwender die Aktivierungsbedingung nicht selbst zusammenstellen muss[6], wird dieses Ereignis bereits beim Initialisieren der Klasse durch die Methode `setEvent` (Listing 14) vorkonfiguriert. Die Überprüfung, ob das konfigurierte Ereignis aufgetreten ist, findet in der `compute:`-Methode der eigenen Klasse (Listing 15). Dafür wird in jedem Tick die Methode `isValid: aSamState` am Ereignisobjekt aufgerufen. Liefert die Methode `false` (das Ereignis ist nicht aufgetreten) zurück, dann macht die Automatik nichts. Bei `true` findet die Kollisionsberechnung statt, die eventuell zur Anzeige eines Warnhinweises führt.

```
1 compute: aSamState
2
3   (event isValid: aSamState)
4     ifFalse: [ ^ aSamState. ].
5
6   ...
7   ^ aSamState
```

Listing 15: Überprüfung des Ereignisauftretens in der Methode `compute:`

6 Das bedeutet nicht, dass der Anwender es nicht machen kann, wenn er es will.

4.5 ANZEIGE DER IDEAL- UND MITTELLINIE

Im letzten Abschnitt dieses Kapitels wird die letzte Klasse von Anzeigen aus Tabelle 2 beschrieben. Es geht dabei um die Konzepte, die entweder die Mittellinie jeder in SAM zu fahrende Strecke oder die Ideallinie, mit der die Versuchspersonen die jeweilige Strecke am schnellsten fahren können[7], farblich hervorheben oder diese in Form einer anderen Anzeige darstellen. Tabelle 7 listet die Klassen auf, die hierfür implementiert wurden.

Tabelle 7: Implementierte Klassen für Anzeige der Ideal- und Mittellinie

Klasse	Team
`AAFDrivingLineAgent` "Fahrlinie hervorheben"	32, 39, 48
`AAFToleranceRangeDisplayAgent` "Toleranzbereich"	29

4.5.1 *Fahrlinie hervorheben*

Die Koordinaten der Mittellinie werden von der ebenfalls im Rahmen des Projekts implementierten Klasse `AAFSupportRacingLine` aus der Kategorie *AAF-Agents-Concepts-Support* bereitgestellt und brauchen von den zu implementierenden Automatiken nicht berechnet werden. Für die Ermittlung der Koordinaten der Mittellinie wurden bei der Implementierung der Automatiken in diesem Abschnitt die folgenden zwei Methoden verwendet

- `racingLine: yy`
- `racingLine: yy Branch: bSymbol.`

Bei der ersten Methode wird als Parameter eine Y-Koordinate (von der Strecke) übergeben, worauf die Methode mit der entsprechenden X-Koordinate der Mittellinie zu diesem Y-Wert antwortet. Diese Methode liefert auch im Falle einer Gabelung nur einen X-Wert zurück und zwar den Wert für den linken Zweig. Braucht man die X-Koordinate für den rechten Zweig oder gar für beide Zweige, dann wird die zweite Methode verwendet. Wenn der zweite Parameter dieser Methode die Zeichenkette `#right` als Wert hat, dann liefert sie die X-Koordinate der Mittellinie für den rechten Zweig; bei Parameterwert `#both` liefert sie zwei X-Werte zurück. Der erste Wert ist der für die Mittellinie des linken Zweiges und der zweite für die Mittellinie des rechten Zweiges.

7 Wie bereits im Abschnitt 3.2 erwähnt wurde, ist die Mittellinie die optimale Linie in SAM und nicht, wie man es erwarten würde, die Ideallinie.

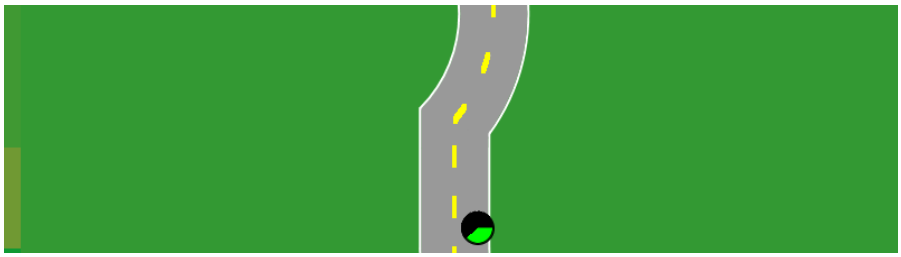

Abbildung 50: Team 39, Färbung der Mittellinie

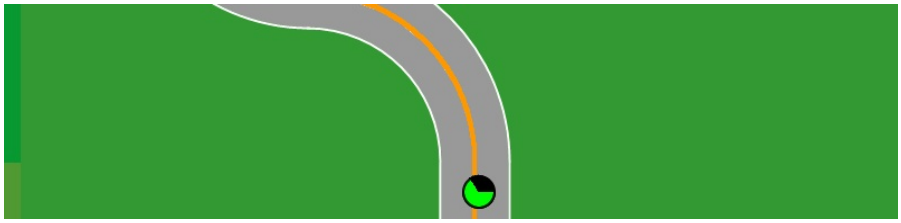

Abbildung 51: Team 48, Färbung der Mittellinie

In Abbildung 50 ist die Anzeige von Team 39 zu sehen. Bei dieser Anzeige wird die ursprünglich weiß und gestrichelt gezeichnete Mittellinie einer Strecke in gelb gefärbt. Auch in der Anzeige von Team 48 (Abbildung 51) wird die Mittellinie gefärbt, aber diesmal in orangener Farbe und als eine durchgängige Linie. Die Mittellinie der beiden Anzeigen wurde mit Hilfe der Methode

- `drawOptimalLine: aSamState forkBranch: branch`

gezeichnet. Der Parameter `branch` bestimmt den zu färbenden Zweig bei einer Gabelung. Das Zeichnen der neuen Mittellinie in der Methode selbst geschieht Pixel für Pixel und nur für den sichtbaren Bereich der Strecke. Auf dem Bildschirm, unabhängig von seiner Größe und Auflösung, ist dieser immer 800 x 768 Pixel groß. Um bestimmen zu können, welcher Teil der Strecke sich aktuell im sichtbaren Bereich befindet, wird zuerst mit Hilfe der bereits bekannten Methode

- `getTrackingObjYPosForCurrentState: aSamState`

die aktuelle Y-Position des Trackingobjekts relativ zum Anfang der Strecke ermittelt. Wir wissen, dass die Y-Koordinate des Mittelpunktes des Objekts auf dem Bildschirm den Wert 730 (vom oberen Bildschirmrand) oder den Wert 38 (vom unteren Rand des sichtbaren Bereichs) hat. Zieht man den zweiten Wert, also die 38 von dem Y-Wert für die aktuelle Objektposition, so bekommt man die Y-Koordinate von dem unteren Rand des sichtbaren Streckenbereichs. Analog kann man 730 (den ersten Wert) zu der Y-Position des Objektes addieren und man bekommt die Y-Koordinate für den oberen Rand des zu sehenden Streckenabschnittes. Hat man den Wert entweder für den unteren oder für den oberen Rand des SAM-Bildschirms, kann man mit der Zeichnung der Mittellinie beginnen. Der Autor hat sich bei

der Implementierung der Methode für den unteren Rand entschieden, deshalb wird in der weiteren Beschreibung der Methode nur dieser erwähnt. Das Zeichnen der Mittellinie findet in einer Zählschleife statt, die bei 0 anfangt und bis 768 (die Größe des sichtbaren Bereichs) hochzählt. Vor dem Beginn der Schleife wird zu dem Y-Wert vom unteren sichtbaren Streckenrand mit Hilfe der Methode `racingLine: yy` oder `racingLine: yy Branch: bSymbol`, je nachdem ob es eine Gabelung ist oder nicht, die X-Koordinate der Mittellinie zu diesem Punkt bestimmt. Danach wird in jedem Schritt der Schleife der Y-Wert um eins inkrementiert und zu diesem neuen Wert wieder die X-Koordinate für die Mittellinie ermittelt. Die so ermittelten Punkte werden in jedem Schritt mit einer geraden Linie verbunden, so dass am Ende der Schleife alle Punkte verbunden eine durchgängige Linie darstellen, die der Mittellinie der Strecke in diesem Abschnitt entspricht. Ist an Stelle der durchgängigen Mittellinie eine gestrichelte gewünscht, so wird in jedem Schritt der Schleife nach der Ermittlung des neuen Punktes geprüft, welche Farbe die Strecke an diesem Punkt hat. Wenn die zurückgelieferte Farbe weiß ist (die Farbe der Standardmittellinie), dann wird der Punkt mit dem vorigen verbunden; wenn sie grau ist, dann wird er übersprungen. Auf diese Weise entsteht die neue gestrichelte Mittellinie.

Die einzige Anzeige, in der statt der Mittellinie die ideale Fahrlinie eingeblendet wird, ist von Team 32. Diese wird mit Hilfe der Methode `drawIdealLine: aSamState forkBranch: branch` gezeichnet. Programmiertechnisch unterscheidet sich das Zeichnen der Ideallinie nur geringfügig von dem der Mittellinie. Dieser besteht darin, dass es in jedem Schritt der Schleife zusätzlich auf das Vorhandensein von Abkürzungen zum aktuellen Y-Wert überprüft wird. Wenn der Test positiv ausfällt, dann werden anstelle der Punkte der Mittellinie diese der Abkürzung ("der optimalen Linie") mit den Punkten der bereits gezeichneten Linie verbunden. Für die Bestimmung der Position und des Verlaufs der Abkürzungen auf einer Strecke werden die Methoden der Klasse `AAFShortCut` und ihrer Unterklassen `AAFShortCut0`, `AAFShortCut1`, `AAFShortCut2`, `AAFShortCut3`[8] aus der Kategorie *AAF-Agents-Concepts-Support* verwendet. Weil die Abkürzungen nicht auf der Mittellinie liegen, aber ihre weiße Farbe, wie im vorigen Absatz bereits beschrieben, zur Erzeugung einer gestrichelten Linie benötigt wird, kann die Ideallinie nur durchgängig und nicht gestrichelt gezeichnet werden.

Konfiguration

Abbildung 52 zeigt den Konfigurationsdialog der Klasse. Eine Beschreibung der einzelnen Parameter ist im Abschnitt C.13 zu finden. Es ist die einzige Klasse, die keine Vorschau anbietet, da der Zeit-

8 Die Klassen wurden im Rahmen der Studien- und Diplomarbeit von Weidner-Kim (2012, in Vorb.) konzipiert und implemenitert.

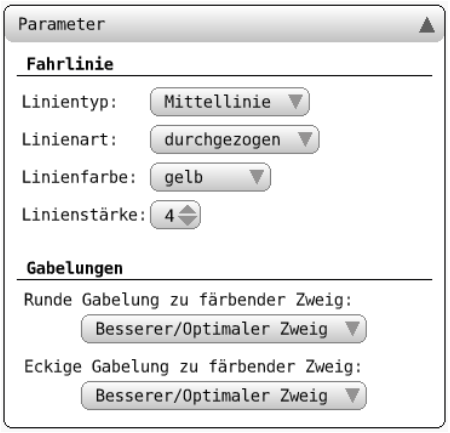

Abbildung 52: Fahrlinie hervorheben: Konfiguration

aufwand für die Implementierung von dieser, wegen der Art der Anzeige und ihrer Parameter, im Vergleich zu den anderen Vorschauen unverhältnismäßig groß ist.

Es sollte darauf hingewiesen werden, dass die Anzeige von dieser Klasse nicht direkt auf den Bilddateien der Strecke gezeichnet wird. Die gewünschte Linie wird stattdessen zuerst auf einer transparenten Ebene gezeichnet und dann über den entsprechenden Streckenabschnitt positioniert. Aus diesem Grund sind links und rechts von der neu gezeichneten Linie Teile der alten weißen Mittellinie zu sehen, wenn die Linienstärke in der Konfiguration auf einen Wert, der kleiner als 4 (die Breite der Standardmittellinie) ist, gesetzt wurde.

4.5.2 *Toleranzbereich*

Die Anzeige der letzten Automatik in diesem Abschnitt ist in Abbildung 53 zu sehen. Bei dieser Anzeige wird am oberen Bildschirmrand die Breite und die Position der zu fahrenden Strecke auf der Höhe der Objektposition angezeigt. Das ist der schmale grüne Bereich in der Mitte der Anzeige. Dieser Bereich wird mit Hilfe der Mittellinie bestimmt. Dafür wird zuerst die aktuelle Y-Position des Trackingobjekts berechnet. Zu diesem Y-Wert wird die X-Koordinate der Mittellinie ermittelt. Ausgehend von diesem Punkt müssen nur noch die Positionen der Fahrbahnbegrenzung links und rechts gefunden werden.

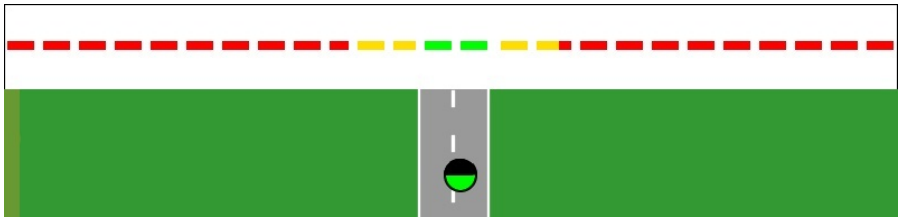

Abbildung 53: Team 29, Anzeige Toleranzbereich

Da die Entfernung der Mittellinie zum Fahrbahnrand nicht immer konstant ist, z. B. bei Kurven ist diese größer als bei geraden Streckenabschnitten, können die Ränder nicht immer durch Addieren und Subtrahieren eines konstanten Wertes bestimmt werden. Daher wurde für die Berechnung der Position der Fahrbahnränder die Methode

- `getTrackBordersPosition: aPoint`

implementiert (Listing 16). Als Parameter wird der Methode ein Punkt von der Mittellinie übergeben. Die Methode sucht von diesem Punkt aus horizontal links und rechts jeweils nach dem ersten Punkt auf dem Streckenbild, der den Farbwert grün hat (das ist die Farbe des Bereichs links und rechts außerhalb der Strecke). Die X-Werte der beiden gefundenen Punkte werden als Ergebnis der Methode zurückgeliefert.

```
1  getTrackBordersPosition: aPoint
2    | borderArray xLeft xRight sensorColor |
3
4    borderArray := Array new: 2.
5    xLeft := (aPoint x) - 15.
6    xRight := (aPoint x) + 15.
7    sensorColor := Display colorAt: (xLeft @ (aPoint y)).
8
9    "determine left border position"
10   [sensorColor = (self class offTrackColor)]
11     whileFalse: [
12       xLeft := xLeft - 2.
13       sensorColor := Display colorAt: (xLeft @ (aPoint y)).
14     ].
15
16   sensorColor := Display colorAt: (xRight @ (aPoint y)).
17
18   "determine right border position"
19   [sensorColor = (self class offTrackColor)]
20     whileFalse: [
21       xRight := xRight + 2.
22       sensorColor := Display colorAt: (xRight @ (aPoint y)).
23     ].
24
25   borderArray at: 1 put: xLeft.
26   borderArray at: 2 put: xRight.
27   ^borderArray.
```

Listing 16: Methode zur Bestimmung der Positionen der Fahrbahnränder

Die zwei Werte, die die oben vorgestellte Methode berechnet, bestimmen die Anfangs- und Endposition des grünen Bereichs in der Automatikanzeige. Links und rechts von diesem Bereich sind zwei gleich große orangefarbene Bereiche zu sehen. Das Konzept bezeichnet diese als Toleranzbereiche. Ihre Aufgabe ist es, die MWB darauf hinzuweisen, dass das von ihnen gesteuerte Objekt sich außerhalb der Strecke

befindet und dass sie es wieder auf die Strecke steuern sollen. Die Breite der Toleranzbereiche wird bei der Konfiguration der Automatik festgelegt. Das Trackingobjekt sollte sich im Idealfall immer im grünen Bereich befinden.

Konfiguration und Vorschau

Die Konfiguration und die Vorschau der letzten in dieser Arbeit implementierten Automatik ist in Abbildung 54 zu sehen. Die einzelnen Parameter werden im Abschnitt C.14 beschrieben.

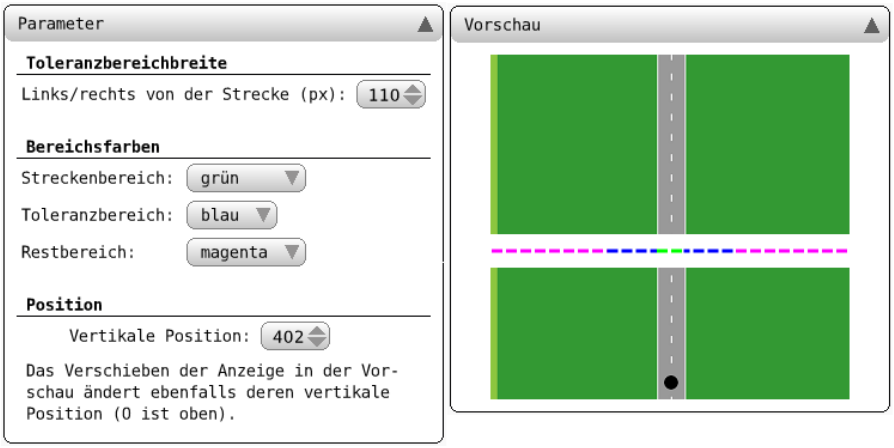

Abbildung 54: Toleranzbereich: Konfiguration und Vorschau

5 TESTS

In diesem Kapitel wird das Testen der in dieser Arbeit umgesetzten Automatiken beschrieben. Dafür wird zuerst im Abschnitt 5.1 kurz das verwendete Testframework vorgestellt. Im Abschnitt 5.2 werden schließlich die durchgeführten Tests aufgelistet und beschrieben.

5.1 TESTFRAMEWORK

Das zum Testen der Automatiken verwendete Testframework wurde in der Diplomarbeit von Kosjar (2012, Abschnitt 7.1) entwickelt. Es basiert auf SUnit[1], das Framework für Unit-Tests in Smalltalk, und besteht aus den Klassen `AAFEventTestCase` (Klassenkategorie *AAF*), `AAFInputProviderAgent` und `AAFTestAgent` (beide in Kategorie *AAF-Agents-Dev*). Die erste Klasse ist die Schnittstelle zum SUnit. Sie stellt die Methoden bereit, die zum Schreiben, Konfigurieren und Starten von Tests gebraucht werden. Die Tests für eine Automatik müssen deshalb in einer Unterklasse von ihr implementiert werden. Wenn ein Test gestartet wird, dann wird auch immer ein SAM-Versuchsschritt gestartet. Damit wird der Ablauf eines Tests visualisiert, was beim Testen von Automatiken mit grafischen Ausgaben sehr praktisch ist. Neben manuellen Tests unterstützt das Framework auch automatisiert ablaufende Tests. Dafür muss beim Erstellen eines Tests die Klasse `AAFInputProviderAgent` verwendet werden. In ihr werden die gewünschten Joystickeingaben für die beiden MWB gespeichert, die später während eines automatisiert ablaufenden Tests anstelle der echten Joystickeingaben in SAM eingespeist werden. Schließlich wird die Klasse `AAFTestAgent` zur Anzeige von weiteren nützlichen Testdaten wie der Name der getesteten Automatik, die Streckenkonfiguration und die Testausgaben der Automatik, wenn es welche gibt, verwendet. Diese werden rechts neben der Streckenanzeige von SAM dargestellt. Die gerade erwähnten Testausgaben einer Automatik können mit Hilfe der von `AAFAgent` geerbten Methode `appendToLog: aString` produziert werden. Neben der Anzeige von diesen, kann das Framework, auf Wunsch des Testentwicklers, nach Ablauf eines Tests diese mit zuvor erstellten Referenzdaten vergleichen. Sind beide Datensätze nicht gleich, dann gilt der Test als nicht erfolgreich und Squeak zeigt eine Fehlermeldung an. Wird dagegen der Test ohne eine Fehlermeldung beendet, dann ist er erfolgreich.

1 http://sunit.sourceforge.net

5.2 TESTS DER AUTOMATIKEN

Das Testen einer Automatik in dieser Arbeit umfasst neben dem Testen der Funktionstüchtigkeit der Klasse selbst auch das Testen ihres Konfigurationsdialogs. Im ersten Fall wird überprüft, ob die Automatik die richtige Anzeige generiert, während im zweiten Fall sichergestellt wird, dass die Parameter einer Automatik, die durch den Anwender eingestellt werden können, vom Konfigurationsdialog richtig gesetzt werden. Diese beiden Fälle werden in den folgenden zwei Unterabschnitten behandelt.

5.2.1 *Agententests*

Für die Automatiken, die im Rahmen dieser Arbeit implementiert wurden, können eigentlich keine Tests geschrieben werden, die basierend auf dem Vergleich der Ist- und Soll-Daten der Automatik entweder erfolgreich sind oder fehlschlagen. Das liegt daran, dass diese Automatiken die Daten, die sie direkt anzeigen oder für die Berechnung ihrer Anzeigen benutzen, nicht selber produzieren sondern von anderen Klassen geliefert bekommen. Folglich bestehen fast alle Tests für eine Automatik nur in der Demonstration ihrer Anzeige. Das ist auch der Grund, warum die hierfür implementierten Klassen den Namen der Automatikklasse tragen, für welche sie die Tests implementieren, gefolgt von der Zeichenkette “Demo” anstelle von “Test”. Die Methoden, in der die Tests realisiert werden, beginnen mit dem Präfix “demo”. Viele der Testklassen haben mehr als eine Demomethode. In jeder von diesen wird die zu demonstrierende Automatik unterschiedlich konfiguriert. Werden alle diese Methoden nacheinander ausgeführt, so werden alle Konfigurationsmöglichkeiten der jeweiligen Automatik gezeigt und “getestet”. In fast jeder Demoklasse gibt es auch eine “echte” Testmethode, deren Name das Präfix “test” hat. In dieser wird die Korrektheit der Ergebnisse von den wenigen Berechnungen, die in der Automatikklasse mit den von den anderen Klassen gelieferten Daten stattfinden, getestet.

Der Aufbau von jeder Testmethode ist gleich. Zuerst wird die zu testende Automatik konfiguriert und dem Testframework übergeben. Darauf folgt die Konfiguration der Strecke und der Hindernisse. Die Daten, welche die Automatiken für ihre Anzeigen benötigen, hängen nicht von der Form und der Länge der Strecke ab, sondern von der Position des Trackingobjekts auf der Strecke und von den Joystickeingaben der beiden MWB. Aus diesem Grund wurden in jeder der implementierten Testklassen für die Konfiguration des Versuchsschrittes die in der Klasse `AAFEventTestCase` voreingestellten Werte übernommen. Es handelt sich dabei um die Strecke “testabschnitt”, die ohne Hindernisse gefahren wird (“obstacleConfig_0”). Diese Strecke ist eine der kleinsten in SAM und kann vom Testframework sehr schnell

geladen werden. Dadurch können die Tests im Vergleich zum Fall mit einer längeren Strecke schneller durchgeführt werden. Die einzigen Ausnahmen sind die Testklassen `AAFDrivingLineAgentDemo` und `AAFDynObstCollisionWarningAgentDemo`. In diesen wird die Strecke "hauptabschnitt_1" verwendet. Sie enthält zum einen die Gabelungen, die die erste Klasse für ihre Tests braucht und zum anderen kann sie zusammen mit der Hinderniskonfiguration "obstacleConfig_1" dynamische Hindernisse anzeigen, die für die Tests der zweiten Klasse benötigt werden. Nachdem der Versuchsschritt konfiguriert wurde, wird der Test gestartet. Nach seinem Ablauf werden die Ist- und Soll-Daten verglichen und der Test wird für erfolgreich bzw. nicht erfolgreich erklärt. Der letzte Schritt kommt nur in einer "echten" Testmethode vor.

Der Rest dieses Unterabschnittes dokumentiert die Testmethoden der Testklassen für alle implementierten Automatiken. Die Methoden die mit der Zeichenkette "demo" beginnen, können auf "Auto", "ManualJoystick" oder "ManualMouse" enden. Bei der ersten Variante handelt es sich um einen automatisiert ablaufenden Test, während bei den letzten beiden der Benutzer über die Joysticks bzw. über die Maus die Steuerung übernehmen muss. Die Methoden mit dem Präfix "test" in ihren Namen laufen immer automatisiert.

AAFJoystickNavigationDisplayAgentDemo

- Methode `demo1[Auto | ManualJoystick | ManualMouse]`
  Testet das Anzeigen des standardmäßig eingestellten rechteckigen Joystickdisplays, auf dem die Ist-Position des Joysticks des linken MWBs (rote, durchgezogene Linie mit einer Kreisspitze), die Ist-Position des Joysticks des rechten MWBs (gelbe, durchgezogene Linie mit einer Kreisspitze) und deren Summe (grüne, gestrichelte Linie mit einer Kreisspitze) visualisiert werden.

- Methode `demo2[Auto | ManualJoystick | ManualMouse]`
  Testet das Anzeigen eines weißen, kreisförmigen Joystickdisplays nur für den linken MWB. Auf diesem werden die Ist-Position des Joysticks des linken MWBs (rote, durchgezogene Linie) und die optimale Joystickposition (schwarze, gestrichelte Linie) visualisiert. Die Anzeige ist auf Position (140, 550).

- Methode `demo3[Auto | ManualJoystick | ManualMouse]`
  Testet das Anzeigen eines Joystickdisplays ohne Hintergrund nur für den rechten MWB. Auf diesem werden die Ist-Position des Joysticks des rechten MWBs (rote, durchgezogene Linie mit einer Pfeilspitze) und die optimale Joystickposition (blaue, gestrichelte Linie mit Pfeilspitze) visualisiert. Die Anzeige befindet sich auf Position (540, 550).

- Methode `testJoystickDisplayValues`
  Testet, ob die Positionswerte korrekt skaliert werden.

AAFSpeedAndControlRegulationArrowsAgentDemo

- Methode `demo[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: ein gelber Pfeil, für den linken MWB auf Position (180, 580), für den rechten MWB auf Position (460, 550).
- Methode `testArrowDisplay`
  Testet, ob die Pfeillänge und -richtung korrekt berechnet werden.

AAFSpeedRegulationArrowsAgentDemo

- Methode `demo1[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: ein schwarzer Pfeil für jede der beiden Personen, die Pfeile sind auf weißem Hintergrund. Anzeige, wenn Geschwindigkeitsabweichung von mehr als 5% vorliegt.
- Methode `demo2[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: ein blauer Pfeil nur für die linke Person, Hintergrund weiß und 50% transparent, Geschwindigkeitsabweichung 10%. Anzeige Y-Position 500.
- Methode `demo3[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: ein roter Pfeil nur für die rechte Person, kein Hintergrund, Geschwindigkeitsabweichung 8%. Anzeige Y-Position 600.
- Methode `testArrowDisplay`
  Testet, ob die Geschwindigkeitsabweichung korrekt berechnet wird.

AAFSpeedRegulationBoxesAgentDemo

- Methode `demo1[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Kästchen für Bremsen und Beschleunigen, Füllfarben (nach aufsteigender Größe - gelb, orange, rot, schwarz) , Rahmenfarbe (schwarz), Transparenz 50%.
- Methode `demo2[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Kästchen nur für Beschleunigen, Füllfarben (nach aufsteigender Größe - blau, orange, cyan, schwarz) , Rahmenfarbe (schwarz), keine Transparenz. Anzeige ist auf Position (110, 470).

- Methode `demo3[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Kästchen nur für Bremsen, Füllfarben (nach aufsteigender Größe - gelb, grün, orange, magenta) , Rahmenfarbe (weiß), keine Transparenz. Anzeige ist auf Position (490, 570).

- Methode `testBoxesDisplay`
  Testet, ob die Kästchen richtig ausgewählt werden.

AAFSteeringProposalDisplayAgentDemo

- Methode `demo1[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Schieberegler für die optimale vertikale und horizontale Joystickposition, Reglerfarbe rot, Indikatorfarbe rot, Transparenz 50%.

- Methode `demo2[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Schieberegler nur für die optimale horizontale Joystickposition, Reglerfarbe blau, Indikatorfarbe gelb, keine Transparenz, Position der Anzeige (230, 670).

- Methode `demo3[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Schieberegler nur für die optimale vertikale Joystickposition, Reglerfarbe orange, Indikatorfarbe rot, keine Transparenz, Position der Anzeige (445, 480).

- Methode `testJoystickDisplayValues`
  Testet, ob die Positionswerte korrekt skaliert werden.

AAFArrowsOnObjectAgentDemo

- Methode `demo1[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: 4 Pfeile am Objekt, Ist-Joystickposition MWB1 (schwarz, gestrichelt), Ist-Joystickposition MWB2 (blau, gestrichelt), Summe der Ist-Positionen (rot, durchgezogen), optimale Joystickposition und -richtung (grün, durchgezogen).

- Methode `demo2[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: nur 1 Pfeil am Objekt, optimale Joystickposition und -richtung (schwarz, durchgezogen).

- Methode `demo3[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: nur 1 Pfeil am Objekt, nur optimale Joystickrichtung (magenta, durchgezogen).

- Methode `testArrowValues`
  Testet, ob die Positionswerte korrekt skaliert werden.

AAFSpeedAndControlRegulationCapsAgentDemo

- Methode `demo[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: 4 Pfeile/Hütchen in cyan.
- Methode `testCapsDisplay`
  Testet, ob die Anzahl der anzuzeigenden Pfeile richtig berechnet wird.

AAFTempomatDisplayAgentDemo

- Methode `demo[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Pfeil nach vorn, Pfeil nach links und Pfeil nach rechts, Pfeilfarbe für Beschleunigen blau, Pfeilfarbe für Bremsen rot, Pfeilfarbe bei optimaler Geschwindigkeit schwarz.
- Methode `testTempomat`
  Testet, ob die Auswahl der Pfeile und ihre Geschwindigkeitsanzeige korrekt ist.

AAFTachometerDisplayAgentDemo

- Methode `demo[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Tachometerfarbe bei optimaler Geschwindigkeit weiß, zulässige Abweichung optimale Geschwindigkeit 8%, Tachometerfarbe in Warnbereich I cyan, zulässige Geschwindigkeitsabweichung für Warnbereich I 15%, Tachometerfarbe in Warnbereich II rot.
- Methode `testTachometer`
  Testet, ob die Geschwindigkeitsabweichung für jeden Bereich korrekt berechnet wird.

AAFColoredObjectAgentDemo

- Methode `demo1[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: das Objekt ist bei 2 Sensoren auf grün in magenta, bei 4 Sensoren auf grün in cyan und bei mehr als 4 Sensoren auf grün in rot gefärbt.
- Methode `demo2[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: das Objekt ist bei 4 Sensoren auf grün in hellgrün, bei 8 Sensoren auf grün in gelb gefärbt. Das Objekt sollte die dritte Farbe (rot) nicht annehmen.

AAFOptimalPosInObjectAgentDemo

- Methode `demo1[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: die optimale Joystickposition ist in magenta, die Geschwindigkeitsanzeige im Objekt ist zu sehen.

- Methode `demo2[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: die optimale Joystickposition ist in rot, die Geschwindigkeitsanzeige im Objekt ist nicht zu sehen, das Objekt ist in blau gefärbt.

- Methode `testOptimalPosInObject`
  Testet, ob die Positionswerte korrekt skaliert werden.

AAFDynObstCollisionWarningAgentDemo

Das dynamische Hindernis ist auf der Strecke für einen sehr kurzen Zeitraum zu sehen. Um dem Beobachter bei der Demonstration der Anzeige trotzdem die Möglichkeit zu geben, sich die angezeigten Hinweise genauer anzuschauen, wurde für diesen Bereich der Strecke die SAM Versuchsschrittdauer von 39ms auf 80ms erhöht.

- Methode `demo1Auto`
  Testet die Automatikanzeige: blinkender Texthinweis für Beschleunigen oder für Bremsen, um eine Kollision mit dem dynamischen Hindernis zu vermeiden.

- Methode `demo2Auto`
  Testet die Automatikanzeige: Pfeilhinweis nur für Beschleunigen, um eine Kollision mit dem dynamischen Hindernis zu vermeiden. Position des Pfeils (100, 470).

- Methode `demo3Auto`
  Testet die Automatikanzeige: transparenter Texthinweis nur für Bremsen, um eine Kollision mit dem dynamischen Hindernis zu vermeiden. Position der Textbox (300, 550).

AAFDrivingLineAgentDemo

- Methode `demo1Auto`
  Testet die Automatikanzeige: durchgängige, orange Mittellinie mit Breite 6px. Bei beiden Arten von Gabelungen (rund und eckig) wird die Linie auf dem linken (schnelleren) Zweig gezeichnet.

- Methode `demo2Auto`
  Testet die Automatikanzeige: gestrichelte, rote Mittellinie mit Breite 4px. Bei der runden Gabelung wird die Linie auf dem rechten und bei der eckigen auf dem linken Zweig gezeichnet.

- Methode `demo3Auto`
  Testet die Automatikanzeige: durchgängige, blaue Ideallinie mit Breite 4px. Bei beiden Arten von Gabelungen (rund und eckig) wird die Linie auf dem linken (schnelleren) Zweig gezeichnet.

AAFToleranceRangeDisplayAgentDemo

- Methode `demo[Auto | ManualJoystick | ManualMouse]`
  Testet die Automatikanzeige: Streckenbereich blau, Toleranzbereich gelb und 80px breit, Restbereich rot, Y-Position der Anzeige 525.
- Methode `testToleranceRange`
  Testet, ob die Streckenränder richtig berechnet werden.

5.2.2 *Tests der Konfigurationsdialogs*

In diesem Abschnitt wird das Testen der Konfigurationsdialoge, die zu jeder Automatik zusätzlich implementiert wurden, vorgestellt. Dieser Schritt soll sicherstellen, dass die Einstellungen, die von einem Benutzer durch die Automatiken-GUI an der Automatik vorgenommen werden, von dieser auch richtig übernommen werden.

Die Tests werden manuell durchgeführt. Hierfür wird zuerst AAFGT gestartet und die Automatik ausgewählt, deren Dialog zu testen ist. Im geöffneten Dialog wird jetzt für jeden Konfigurationsparameter jeder seiner Werte ausgewählt und auf die Änderung der Vorschau geachtet (Abbildung 55).

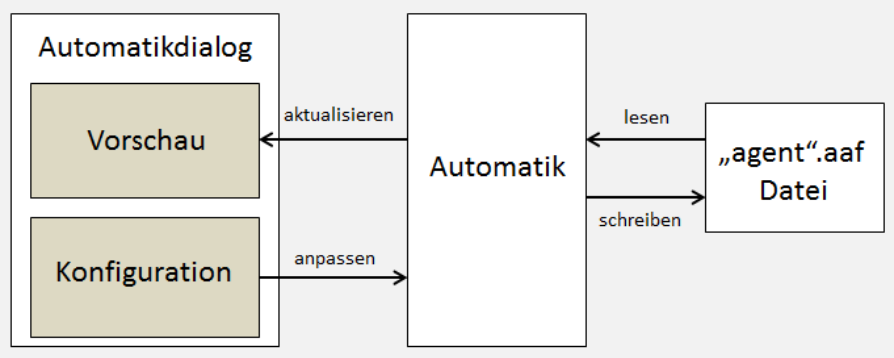

Abbildung 55: Konfiguration von Automatiken

Ändert sich die Vorschau nicht oder die Änderung entspricht nicht dem, was erwartet wird, dann ist der Dialog oder die Vorschau fehlerhaft. Um herauszufinden, welche der beiden Komponenten den Fehler enthält, wird die Automatik in AAFGT gespeichert. Dadurch werden alle ihre Parameter in die für die Speicherung angegebene Datei geschrieben. Wenn die gespeicherten Werte der in der Konfiguration gesetzten entsprechen, dann liegt der Fehler in der Vorschau. Ansonsten ist der Fehler in der Konfiguration zu suchen. Nachdem dieser behoben wurde, beginnt das Testen von neuem.

6 BEWERTUNG

In diesem Kapitel wird der Versuch unternommen, die im Rahmen dieser Arbeit implementierten Automatikfunktionen hinsichtlich ihrer Anzeigen zu bewerten. Dabei ist das Ziel der Bewertung nicht die Bestimmung der Automatik mit der besten Anzeige, sondern vielmehr die Ermittlung einer optimalen Konfiguration für die jeweilige Anzeige.

Die im Kapitel 3 vorgestellten und im Kapitel 4 implementierten Automatikanzeigen haben eine Hinweisfunktion, d.h. diese visualisieren bestimmte Größen innerhalb des SAM-Prozesses und sind als Unterstützung für die beiden MWB gedacht. Hierfür erfolgt keinerlei Interaktion zwischen den Personen und der Anzeige wie dies bei sogenannten Dialogfenstern der Fall ist. Während es für die letztgenannten grafischen Elemente eine große Auswahl an Literatur gibt, die sich mit der Gestaltung und der Bewertung von dieser Art von Anzeigen beschäftigt, existieren für die in dieser Arbeit vorkommenden Anzeigen kaum solche Dokumente. Eins von diesen ist die Norm ISO 9241-12 von CEN (1998). Dieser 12. Teil von ISO 9241 "befasst sich mit der visuellen Informationsdarstellung bei der Verwendung von Bildschirmgeräten". Neben ergonomische Empfehlungen für die Informationsdarstellung für zeichenorientierte oder grafische Mensch-Maschine-Schnittstellen, werden in diesem Teil der Norm auch die Eigenschaften der dargestellten Information spezifiziert. Die Norm kann aus diesem Grund sowohl für die Entwicklung als auch für die Bewertung von visueller Informationsdarstellung verwendet werden.

6.1 EIGENSCHAFTEN VON ANZEIGEN

Laut der oben kurz vorgestellten Norm ISO 9241-12 sollte die dargestellte Information folgende charakteristische Eigenschaften besitzen:

- *Klarheit*
  Der Informationsinhalt wird schnell und genau vermittelt.
- *Unterscheidbarkeit*
  Die angezeigte Information kann genau unterschieden werden.
- *Kompaktheit*
  Den Benutzern wird nur die relevante Information gegeben.
- *Konsistenz*
  Die gleiche Information wird entsprechend den Erwartungen des Benutzers stets auf gleiche Art dargestellt.

- *Erkennbarkeit*
  Die Aufmerksamkeit des Benutzers wird zur benötigten Information gelenkt.

- *Lesbarkeit*
  Die Information ist leicht zu lesen.

- *Verständlichkeit*
  Die Bedeutung ist leicht verständlich, eindeutig, interpretierbar und erkennbar.

Besitzt eine Anzeige diese Eigenschaften, dann wird sie vom Benutzer besser wahrgenommen und dieser kann seine Arbeit effektiv und effizient ausführen. Das Gestaltungsziel für die Anzeige sollte deshalb immer das Erreichen dieser Eigenschaften sein[1].

6.2 EMPFEHLUNGEN ZUR GESTALTUNG DER BILDSCHIRMDARSTELLUNG

Dieser Abschnitt stellt die Empfehlungen vor, die die Norm für die visuelle Darstellung von Informationen auf dem Bildschirm macht. Diese sind vor allem für Entwickler von solchen Anzeigen interessant. Dabei werden nicht alle in der Norm enthaltenen Empfehlungen beschrieben, sondern nur diejenigen, die aus der Sicht des Autors für die Anzeigen in dieser Arbeit relevant und anwendbar[2] sind. Dafür wurden wiederholt die Konzeptbögen analysiert. Die Empfehlungen sind in den Kategorien *Organisation der Information*, *Grafische Objekte* und *Kodierverfahren* unterteilt.

Organisation der Information

- *Platzierung der Information*
  Die Information sollte dort platziert werden, wo es den Erwartungen der Benutzer und den Anforderungen der Arbeitsaufgabe entspricht.

  BEISPIEL: Die Anzeige der Joystickposition für den linken MWB sollte auf der linken SAM-Bildschirmhälfte positioniert werden.

- *Dichte der angezeigten Informationen*
  Die Dichte der angezeigten Informationen sollte derart sein, dass der Benutzer sie nicht als überfüllt empfindet.

  ANMERKUNG: Bei grafischen Mensch-Maschine-Schnittstellen können andere Elemente, wie Linien oder Bildschirmsymbole, den Eindruck einer höheren Informationsdichte hervorrufen.

1 vgl. CEN (1998), S. 7
2 Die Norm enthält Empfehlungen auch für die Bildschirmdarstellung von Dialogfenstern, die für diese Arbeit nicht von Interesse sind.

- *Gesetz der Ähnlichkeit*
  Elemente werden als zusammengehörend wahrgenommen, wenn sie einander ähnlich sind.

  BEISPIEL: Wenn als Hinweis zur Beschleunigung ein Pfeil benutzt wird, dann sollte der Hinweis zum Bremsen auch ein Pfeil sein und z.B. keine Textbox.

Grafische Objekte

- *Statusunterscheidung bei grafischen Objekten*
  Um den unterschiedlichen Status von grafischen Objekten anzuzeigen, sollten Kodierverfahren verwendet werden.

  BEISPIEL: Das aktive Element einer Anzeige wird durch eine andere Form oder Farbe von den restlichen Elementen unterschieden.

- *Unterscheidung von Objekten gleicher Art*
  Werden identische grafische Darstellungen (Bildschirmsymbole) für unterschiedliche Objekte verwendet, dann sollte jede Darstellung durch eine Beschriftung eine eindeutige Identifikation erhalten.

Kodierverfahren

- *Grafische Kodierung und Abstufungen*
  Die Anzahl der zu unterscheiden Abstufungen oder Grade der Kodierung sollte begrenzt sein.

- *Gestaltung von Bildschirmsymbolen*
  Bildschirmsymbole sollten solcherart gestaltet sein, dass sie leicht erkannt und unterschieden werden können. Sie sollten leicht und klar verständlich sein.

- *Kodierung mittels Linien*
  Wird eine Kodierung mittels verschiedenartiger Linien verwendet, so sollten die Unterschiede der Linienart (z.B. durchgezogen, gestrichelt, punktiert) und der Linienstärke (normal, fett) deutlich voneinander unterscheidbar sein.

- *Richtung von Linien*
  Wird die Richtung von Linien zur Kodierung einer Richtung oder eines Wertes verwendet, dann sollte ausreichende Kontextinformation vorgesehen werden, damit die Richtung oder der Wert genau identifizierbar ist.

- *Farbe als Hilfskodierung*
  Farbe sollte nie das einzige Mittel zur Kodierung sein, denn

machen Menschen können bestimmte Farben schwer oder überhaupt nicht voneinander unterscheiden. Farbe ist ein guter Hilfskode. Sie sollte zusammen mit anderen Kodierverfahren redundant verwendet werden.

- *Angemessene Verwendung von Farbe*
  Willkürliche Verwendung von Farbe sollte vermieden werden, da dies den Eindruck eines "vollen" oder "zugepackten" Bildschirms erwecken und die Effektivität der Farbkodierung in anderen Situationen vermindern kann.

- *Zugehörigkeit von Farbe zu Informationskategorien*
  Wird Farbe als vorherrschender Kode verwendet, dann sollte jede Farbe nur eine Informationskategorie versinnbildlichen. Wird dieselbe Farbe für verschiedene Informationskategorien verwendet, kann dies das Erkennen der beabsichtigten Bedeutung durch den Benutzer behindern.

  BEISPIEL: Die Farbe Rot sollte in allen Anzeigen eine Verringerung der Geschwindigkeit oder Bremsen signalisieren.

- *Konventionen für Farbkodierung*
  Gebräuchliche Konventionen für Farbkodierung sollten unter Berücksichtigung des Nutzungskontextes eingehalten werden (z.B. rot = Gefahr, gelb = Vorsicht, grün = in Ordnung oder verfügbar). Außerdem sollte die Verwendung von Farbe konsistent mit Konventionen der Arbeitsaufgabe und kulturellen Konventionen sein.

- *Anzahl der verwendeten Farbe*
  Wird Farbkodierung verwendet, dann sollten die Farben für den Benutzer leicht unterscheidbar sein. Es ist zu empfehlen, außer Schwarz und Weiß nicht mehr als sechs Farben zu verwenden.

- *Gesättigtes Blau*
  Gesättigtes Blau sollte für die Anzeige von Text oder Symbolen vor einem dunklen Hintergrund vermieden werden (bei kleinen Elementen in gesättigtem Blau ist es meist schwierig, diese zuverlässig zu untersuchen und das Auge richtig zu akkommodieren).

- *Chromostereopsis*
  Farben hoher Sättigung mit spektral sehr unterschiedlichen Wellenlängen (wie Rot und Blau), die ungewollte Tiefeneffekte oder übermäßige Akkommodation hervorrufen, sollten bei Leseaufgaben nicht nebeneinander als Text oder Hintergrund verwendet werden.

- *Vordergrundfarben*
  Werden vor neutralem Hintergrund (d.h. Weiß, Grau, Schwarz)

Vordergrundfarben verwendet, dann sollten Vordergrundfarben gewählt werden, die auf dem 1976 CIE UCS Farbdiagramm weit auseinander liegen, um die Fähigkeit des Benutzers zu verbessern, die Farben zu unterscheiden.

BEISPIEL: Hellgelb wird zusammen mit Blau verwendet.

- *Hintergrundfarben*
  Farben hoher Sättigung (und helles Weiß) sollten als Hintergrundfarbe vermieden werden.

  ANMERKUNG: Eine gute Hintergrundfarbe ist z.B. helles Grau.

- *Kodierung durch Blinken*
  Wird Blinken als Kodierung verwendet, sollte es für Anwendungen in Betracht gezogen werden, in denen ein angezeigtes Element eine wichtige Bedeutung für die Aufmerksamkeit des Benutzers hat.

- *Hervorheben durch Blinken*
  Ist ein Hervorheben durch Blinken beabsichtigt und ist das Lesen des Textes wichtig, dann sollte besser ein anderes Verfahren zur Hervorhebung der Information in Betracht gezogen werden.

  BEISPIEL: Ein Symbol wird hinzugefügt, um den Text zu markieren, und das Symbol blinkt anstelle des Textes. Dieses Verfahren zieht die Aufmerksamkeit auf sich, ohne die Lesbarkeit zu beeinträchtigen.

- *Kodierung von Flächen*
  Müssen in Diagrammen Flächen unterschieden werden, dann sollte anstelle einer Farbkodierung das Füllen der Flächen mit unterschiedlichen Mustern (Schraffur, Schattierung, Punktieren) in Betracht gezogen werden. Kodierung durch Textur sollte auch in Verbindung mit Farbkodierung in Betracht gezogen werden, um eine redundante Kodierung zu erreichen.

An dieser Stelle sei darauf hingewiesen, dass die oben aufgelisteten Empfehlungen in ihrer Gesamtheit nicht für die Anzeige von jeder einzelnen in dieser Arbeit implementierten Automatik anwendbar sind. Welche von ihnen für welche Anzeige relevant ist, ist in Anhang B nachzulesen.

6.3 BEWERTUNG

Die Bewertung der Anzeigen der im Rahmen dieser Arbeit implementierten Automatiken basiert auf dem Verfahren in Anhang A der Norm ISO 9241-12. Unter Verwendung von diesem kann überprüft werden, ob die von der Norm gegebenen Empfehlungen erfüllt sind.

Hierfür muss zuerst bestimmt werden, welche Empfehlungen für welche Anzeige anwendbar sind. Dieser Schritt wurde bereits im Abschnitt 6.2 behandelt. Im nächsten Schritt wird untersucht, ob diese relevanten Empfehlungen erfüllt werden. Das Beurteilen der Einhaltung erfolgt unter Verwendung von einer oder mehreren der folgenden Methoden[3]:

1. *Messungen*
   Messungen beziehen sich auf das Messen oder Berechnen einer Größe, welche die Informationsdarstellung betrifft.

2. *Inspektion*
   Bei dieser Methode wird die Informationsdarstellung geprüft oder inspiziert, um zu bestätigen, dass eine bestimmte feststellbare Eigenschaft zutrifft.

3. *Nachweis durch Dokumente*
   Diese Methode bezieht sich auf jegliche relevante Information über die Informationsdarstellung, soweit daraus die Einhaltung der entsprechenden Empfehlungen hervorgehen.

4. *analytische Bewertung*
   Die analytische Bewertung bezieht sich auf die auf Erfahrung beruhende Beurteilung der Informationsdarstellung durch einen Fachmann dieses Gebiets.

5. *empirische Bewertung*
   Diese Art der Bewertung bezieht sich auf die Anwendung von Prüfverfahren unter Einbeziehung repräsentativer Endbenutzer, um die Einhaltung einer Empfehlung zu untersuchen.

Für die Überprüfung der Einhaltung der entsprechenden Empfehlungen wird in dieser Arbeit ausschließlich die Methode der Inspektion angewendet, da die Anzeigen hierfür bereits implementiert wurden. Das Ergebnis der Überprüfung ist Anhang B zu entnehmen. Die optimale oder empfohlene Konfiguration für den jeweiligen Agenten, die auf diesem Ergebnis basiert, ist in Anhang C zu finden. Dort werden alle Konfigurationsparameter und ihre zulässigen Werte aufgelistet. Die ermittelten, optimalen Werte stellen dabei die Standardwerte für die Parameter.

3 vgl. CEN (1998), S. 18

7 ZUSAMMENFASSUNG UND AUSBLICK

7.1 ZUSAMMENFASSUNG

Das Ziel der vorliegenden Arbeit war die Implementierung von Automatiken und Funktionen für SAM, die die aktuelle und optimale Objektposition, -richtung und -geschwindigkeit und/oder die aktuelle und optimale Position der Joysticks der beiden MWB anzeigen. Diese Automatiken dienen nur der Unterstützung der beiden Personen und greifen nicht in deren Steuerung ein. Die Implementierung basiert auf Konzepten, die im Vorfeld dieser Arbeit im Rahmen der Dissertation von Kain entstanden sind. Nach der Analyse von diesen in Kapitel 3 wurde in Kapitel 4 die Umsetzung von insgesamt 14 Klassen (Anhang A) in Squeak beschrieben. In diesen konnten alle aus der Analyse resultierenden Automatiken und Funktionen realisiert werden. Zu jeder der Klassen wurde zusätzlich ein Konfigurationsdialog implementiert, den die Anwender nutzen können, um die Automatik an ihre Wünsche anzupassen. Dabei wurden auch Konfigurationsparameter hinzugefügt, die in den entsprechenden Konzepten nicht vorgesehen waren. Zur Überprüfung der Korrektheit der Anzeigen wurden Squeak-Klassen implementiert, die sowohl Methoden zum Testen der Anzeigen als auch zur Demonstration von diesen enthalten. In Kapitel 6 wurde zum Schluss der Versuch unternommen, die implementierten Automatikanzeigen zu bewerten, mit dem Ziel für jede Automatik die beste Konfiguration zu ermitteln. Für die Bewertung wurde das Verfahren aus der Norm ISO 9241-12 verwendet.

Für die Berechnung der optimalen Joystick- und Objektpositionen, die durch die Automatiken dieser Arbeit angezeigt werden, wurden die Methoden der Klassen aus der Studien- und Diplomarbeit von Weidner-Kim (2012, in Vorb.) benutzt. Die Diplomarbeiten von Kosjar (2012) und Wickert (2012b, in Vorb.) sind zwei weitere Arbeiten, die sich mit der Umsetzung von Automatiken für das Projekt ATEO beschäftigen.

7.2 AUSBLICK

Die Implementierung der in diesem Dokument behandelten Automatiken kann als abgeschlossen angesehen werden. Es könnten eventuell bei der einen oder anderen ein weiterer Konfigurationsparameter hinzugefügt werden, aber das sollte nicht mehr zu Änderungen der Anzeige selbst führen. Sollen in Zukunft weitere Anzeigen implementiert werden, die auf den bereits vorhandenen basieren, dann kann

das am Besten durch das Erstellen von neuen Klassen erledigt werden, die die geeigneten Klassen durch Vererbung erweitern. Dabei sollte der Entwickler versuchen, so viele Teile der Anzeige wie möglich mit Hilfe von bereitgestellten Squeak-Funktionen zu realisieren. Dadurch können diese leichter oder gar erst parametrisiert werden.

Ein großer Nachteil der hier implementierten Anzeigen ist ihre Dynamik. Ihr Inhalt ändert sich so schnell, dass die MWB mit ihren Joystickbewegungen ihr gar nicht folgen können. Der Grund hierfür liegt in SAM, wo jede Abweichung des Objektmittelpunkts von der Mittellinie der Strecke als Fehler berechnet wird. Weil die MWB das Objekt nach Möglichkeit mit maximaler Geschwindigkeit steuern sollen und ihre Reaktion als Menschen verzögert ist, ist es unmöglich, das Objekt so zu fahren, dass es keine Fehler produziert und weil die Berechnung der optimalen Lenkung in jedem Tick auch auf diesem Algorithmus basiert, "dreht" die Anzeige "durch". Eine mögliche Lösung wäre, die Methode zur Berechnung des Flächenfehlers so umzuschreiben, dass die Fehlerberechnung erst dann beginnt, wenn das Objekt den grauen Bereich der Strecke verlässt. Eine weitere Möglichkeit das Problem zu entschärfen besteht auf der Seite der Anzeige. Anstatt nur die optimale Auslenkung für den nächsten Tick anzuzeigen, könnten hier die optimalen Positionen für die nächsten x Ticks berechnet werden, woraus eine Tendenz abgeleitet werden kann, die dann schließlich angezeigt wird. Dies könnte zu sanfteren Übergängen in der Anzeige führen. Ob diese beiden Ansätze zu einer verbesserten Anzeige führen, müsste durch empirische Untersuchungen mit Probanden im Labor ermittelt werden, nachdem sie z.B. im Rahmen weiterer studentischen Arbeiten umgesetzt wurden.

LITERATURVERZEICHNIS

[Black u. a. 2007] Black, Andrew P. ; Ducasse, Stephane ; Nierstrasz, Oscar ; Pollet, Damien: *Squeak by Example*. 2009. Square Bracket Associates, Switzerland, 2007. – URL http://squeakbyexample.org. – ISBN 978-3-9523341-0-2

[Bothe u. a. 2010] Bothe, Klaus ; Hildebrandt, Michael ; Niestroj, Nicolas: *ATEO-System Komponente: SAMs*, 2010. – URL https://www2.informatik.hu-berlin.de/swt/lehre/MTI/ressources/Softwarespezifikation_SAMs.pdf. – Abruf 2010

[CEN 1998] CEN: *EN ISO 9241-12, Ergonomische Anforderung für Bürotätigkeiten mit Bildschirmgeräten, Teil 12: Informationsdarstellung*. Europäisches Komitee für Normung, Brüssel, Belgien. Dezember 1998. – Deutsche Fassung

[Fuhrmann 2010] Fuhrmann, Esther: *Entwicklung eines GUI für die Konfiguration der Software-Komponente zur Systemprozessüberwachung und -kontrolle in einer psychologischen Versuchsumgebung*, Humboldt-Universität zu Berlin, Diplomarbeit, 2010

[Goldberg und Robson 1983] Goldberg, Adele ; Robson, David: *Smalltalk-80: the language and its implementation*. Addison-Wesley Publishing Company, 1983. – ISBN 0-201-11371-6

[Goldberg u. a. 1983] Goldberg, Adele ; Robson, David ; Krasner, Glenn ; Krasner, Glenn (Hrsg.): *Smalltalk-80: bits of history, words of advice*. Addison-Wesley Publishing Company, 1983. – ISBN 0-201-11669-3

[Hasselmann 2012] Hasselmann, Michael: *Erweiterung einer Softwarekomponente für Automatiken zur Systemprozessüberwachung und -führung als Bestanteil einer psychologischen Versuchsumgebung*. 2012. – Diplomarbeit in Bearbeitung

[Kain 2012] Kain, Saskia: *Entwickler in komplexen Mensch-Maschine-Systemen: Analyse des Einflusses von Entwicklerressourcen auf den Entwicklungsprozess und das Ergebnis*. 2012. – Dissertation in Vorbereitung

[Kosjar 2011] Kosjar, Nikolai: *Die Gebrauchstauglichkeit des Automatiken-GUI im Projekt Arbeitsteilung Entwickler Operateur (ATEO)*, Humboldt-Universität zu Berlin, Studienarbeit, 2011

[Kosjar 2012] Kosjar, Nikolai: *Ein Ereignis-System für das ATEO Automation Framework sowie die Implementierung und Testung von au-*

ditiven und visuellen Hinweisen, Humboldt-Universität zu Berlin, Diplomarbeit, 2012

[Leonhard 2012] LEONHARD, Christian: *Fenster zum Prozess: Weiterentwicklung eines Operateursarbeitsplatzes im Projekt Arbeitsteilung Entwickler Operateur (ATEO)*, Humboldt-Universität zu Berlin, Diplomarbeit, 2012

[Seid 2012] SEID, Aydan: *Erweitern der Log-Datei und der Analyse von Log-Dateien im ATEO Projekt*, Humboldt-Universität zu Berlin, Studienarbeit, 2012

[Weidner-Kim 2012] WEIDNER-KIM, Helmut: *Konzeption von Algorithmen für die Implementierung von Automatiken zur Bestimmung des Ist-Prozessverlaufs und Vorgabe des Soll-Prozessverlaufs*. 2012. – Studien- und Diplomarbeit in Vorbereitung

[Wickert 2012a] WICKERT, Andreas: *Fehler und deren Vermeidung bei der Programmierung und Konfiguraiton von Autoamtiken im ATEO-Projekt*, Humboldt-Universität zu Berlin, Studienarbeit, 2012

[Wickert 2012b] WICKERT, Andreas: *Klassifizierung, Umsetzung und Testung von Automatiken für regelnde Eingriffe in die Objektsteuerung von SAM*. 2012. – Diplomarbeit in Vorbereitung

ANHANG

A

LISTE DER KLASSEN

Die Tabelle unten listet die Namen der in dieser Arbeit implementierten Automatikklassen. Auf ein Klassendiagramm wurde verzichtet, weil alle Klassen voneinander unabhängig sind und es somit keine Klassenhierarchie existiert, die es abzubilden gibt. Der Name der Automatik in Squeak ist in jeder Zeile der folgenden Tabelle in `nicht-proportionaler` Schrift und ihr Name in AAFGT in normaler Schrift dargestellt.

#	Name
1	`AAFJoystickNavigationDisplayAgent` Joystick-Navigationsanzeige
2	`AAFSpeedAndControlRegulationArrowsAgent` Steuerungsregulierung
3	`AAFSpeedRegulationArrowsAgent` Geschwindigkeitsregulierung (Pfeil)
4	`AAFSpeedRegulationBoxesAgent` Geschwindigkeitsregulierung (Kästchen)
5	`AAFSteeringProposalDisplayAgent` Steuerungsvorgabenanzeige
6	`AAFArrowsOnObjectAgent` Pfeile am Objekt
7	`AAFSpeedAndControlRegulationCapsAgent` Steuerungsregulierung am Objekt
8	`AAFTempomatDisplayAgent` Tempomat-Anzeige
9	`AAFTachometerDisplayAgent` Tachometer-Anzeige
10	`AAFColoredObjectAgent` Farbänderung Objekt
11	`AAFOptimalPosInObjectAgent` Joystickanzeige im Objekt
12	`AAFDynObstCollisionWarningAgent` Hindernis-Kollisionswarnhinweis (dynamisch)

13	`AAFDrivingLineAgent` Fahrlinie hervorheben
14	`AAFToleranceRangeDisplayAgent` Toleranzbereich

B

BEWERTUNG DER AUTOMATIKANZEIGEN

In diesem Anhang findet die Bewertung der Automatikanzeigen gemäß dem im Kapitel 6 vorgestellten Verfahren statt. Dafür wird in jedem der folgenden Abschnitte eine Tabelle mit den zu der Anzeige passenden Empfehlungen erstellt und überprüft, ob diese erfüllt werden. Basierend darauf wird versucht, die optimale Konfiguration für die jeweilige Automatikanzeige zu ermitteln. Zu jedem Abschnitt in diesem Anhang gibt es einen Abschnitt im Anhang C, der denselben Namen hat und die Konfigurationsparameter für die Anzeige beschreibt. Die Konfigurationswerte, die dort als Standardwerte gekennzeichnet sind, entsprechen den hier ermittelten optimalen Konfigurationswerten.

B.1 JOYSTICK-NAVIGATIONSANZEIGE

Empfehlungen	J / N	Kommentare
Platzierung der Information	J	Optimal auf der Bildschirmhälfte der Person etwas über der Höhe des Trackingobjekts
Dichte der angezeigten Information	J	Max. 3 Positionen anzeigen (4 ist schon kritisch)
Statusunterscheidung bei grafischen Objekten	J	Unterscheidung durch Verwendung von unterschiedlicher Form und Farbe für die Positionsanzeigen
Unterscheidung von Objekten gleicher Art	N	Keine Beschriftung für die einzelnen Positionsanzeigen
Gestaltung von Bildschirmsymbolen	J	Leicht zu verstehen
Kodierung mittels Linien	J	Gestrichelte und durchgezogene Linien
Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung

Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Konventionen für Farbkodierung	N	Rot, gelb, grün dienen nur der Unterscheidung der Anzeigen
Anzahl der verwendeten Farbe	J	9 Farben (weiß, schwarz, rot, grün, gelb, blau, cyan, magenta, orange) zur Auswahl
Gesättigtes Blau	J	Kein dunkler Hintergrund
Vordergrundfarben	J	
Hintergrundfarben	J	Optimal helles Grau

B.2 STEUERUNGSREGULIERUNG

EMPFEHLUNGEN	J / N	KOMMENTARE
Platzierung der Information	J	Optimal auf der Bildschirmhälfte der Person etwas über der Höhe des Trackingobjekts
Dichte der angezeigten Information	J	Anzeige nur eines Pfeils
Gestaltung von Bildschirmsymbolen	J	Leicht zu verstehen
Farbe als Hilfskodierung	J	Als zusätzliche Kodierung
Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Konventionen für Farbkodierung	N	Die Farbe dient nur der Hervorhebung der Anzeige
Anzahl der verwendeten Farbe	J	7 Farben (schwarz, rot, gelb, blau, cyan, magenta, orange) zur Auswahl, eine wird verwendet
Gesättigtes Blau	N	Blau auf gesättigtem Grün nicht zu empfehlen
Vordergrundfarben	J	Optimal - alle bis auf blau

Hintergrundfarben	N	Grün ist vorgegeben

B.3 GESCHWINDIGKEITSREGULIERUNG (PFEIL)

Empfehlungen	J / N	Kommentare
Platzierung der Information	J	Optimal auf der Bildschirmhälfte der Person etwas über der Höhe des Trackingobjekts
Dichte der angezeigten Information	J	Ein Pfeil links und ein Pfeil rechts
Unterscheidung von Objekten gleicher Art	N	Keine Beschriftung für die einzelnen Pfeile
Gestaltung von Bildschirmsymbolen	J	Leicht zu verstehen
Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung
Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Konventionen für Farbkodierung	N	Farbe dient nur der Hervorhebung der Anzeige
Anzahl der verwendeten Farbe	J	8 Farben (schwarz, rot, grün, gelb, blau, cyan, magenta, orange) zur Auswahl, eine wird verwendet
Gesättigtes Blau	J	Kein dunkler Hintergrund (weiß)
Vordergrundfarben	J	Auf weiß leicht unterscheidbare Farben
Hintergrundfarben	N	Helles Weiß (mit 50% Transparenz besser)

B.4 GESCHWINDIGKEITSREGULIERUNG (KÄSTCHEN)

EMPFEHLUNGEN	J / N	KOMMENTARE
Platzierung der Information	J	Optimale Position - horizontal mittig auf dem Bildschirm, vertikal etwas über der Hälfte des Bildschirm, um die Hindernisse nicht zu verdecken
Dichte der angezeigten Information	J	Hinweis zur Geschwindigkeitsänderung
Statusunterscheidung bei grafischen Objekten	J	Unterscheidung durch unterschiedliche Färbung des aktiven Kästchens
Unterscheidung von Objekten gleicher Art	N	Keine Beschriftung für die Kästchen für Beschleunigen und für Bremsen
Grafische Kodierung und Abstufungen	J	4 Kästchen für Beschleunigen und 4 für Bremsen
Gestaltung von Bildschirmsymbolen	J	Leicht zu verstehen
Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung
Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Zugehörigkeit von Farbe zu Informationskategorien	J	Z.B. das zweite Kästchen für Beschleunigen hat dieselbe Farbe und Intensität wie das zweite Kästchen für Bremsen
Konventionen für Farbkodierung	N	Farbe dient nur der Hervorhebung der Anzeige
Anzahl der verwendeten Farbe	J	9 Farben (weiß, schwarz, rot, grün, gelb, blau, cyan, magenta, orange) zur Auswahl, 4 werden verwendet

Gesättigtes Blau	N	Blau auf gesättigtem Grün nicht zu empfehlen
Vordergrundfarben	J	Leicht zu unterscheiden
Hintergrundfarben	N	Grün ist vorgegeben
Kodierung von Flächen	N	Statt mit Farbe, könnten die Kästchen mit unterschiedlichen Mustern gefüllt werden

B.5 STEUERUNGSVORGABENANZEIGE

EMPFEHLUNGEN	J / N	KOMMENTARE
Platzierung der Information	J	Optimal - horizontale Anzeige mittig unter dem Trackingobjekt, vertikale Anzeige horizontal mittig und vertikal etwas über der Höhe des Objekts
Dichte der angezeigten Information	J	Horizontale und vertikale Joystickposition
Gesetz der Ähnlichkeit	J	Beide Anzeigen sind ähnlich und erwecken den Eindruck, dass sie zusammengehören
Statusunterscheidung bei grafischen Objekten	J	Das aktive Element ist in einer anderen Farbe
Unterscheidung von Objekten gleicher Art	N	Keine Beschriftung für die einzelnen Positionsanzeigen
Gestaltung von Bildschirmsymbolen	J	Leicht zu verstehen
Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung
Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Konventionen für Farbkodierung	N	Farbe dient nur der Hervorhebung der Anzeige

Anzahl der verwendeten Farbe	J	7 Farben (rot, grün, gelb, blau, cyan, magenta, orange) zur Auswahl
Gesättigtes Blau	N	Blau auf gesättigtem Grün nicht zu empfehlen
Vordergrundfarben	J	Leicht zu unterscheiden vom Hintergrund
Hintergrundfarben	N	Grün ist vorgegeben

B.6 PFEILE AM OBJEKT

Empfehlungen	J / N	Kommentare
Platzierung der Information	J	Direkt am Objekt
Dichte der angezeigten Information	N	4 (wenn alle eingeschaltet) sich schnell in Richtung und Länge ändernde Pfeile
Statusunterscheidung bei grafischen Objekten	J	Unterscheidung durch Verwendung von unterschiedlicher Farbe für die einzelnen Pfeile
Unterscheidung von Objekten gleicher Art	N	Keine Beschriftung für die einzelnen Pfeile
Gestaltung von Bildschirmsymbolen	N	Leicht zu verstehen, aber schwer zu unterscheiden
Kodierung mittels Linien	J	Gestrichelte und durchgezogene Linien
Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung
Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Zugehörigkeit von Farbe zu Informationskategorien	J	Unterschiedliche Farbe für die unterschiedlichen Pfeile
Konventionen für Farbkodierung	N	Farbe dient nur der Hervorhebung der Anzeige

Anzahl der verwendeten Farbe	J	8 Farben (schwarz, rot, grün, gelb, blau, cyan, magenta, orange) zur Auswahl
Gesättigtes Blau	N	Blau auf gesättigtem Grün nicht zu empfehlen
Vordergrundfarben	J	Leicht zu unterscheiden
Hintergrundfarben	N	Grün ist vorgegeben

B.7 STEUERUNGSREGULIERUNG AM OBJEKT

EMPFEHLUNGEN	J / N	KOMMENTARE
Platzierung der Information	J	Direkt am Objekt
Dichte der angezeigten Information	J	Anzeige der Positionsabweichung
Grafische Kodierung und Abstufungen	J	Min. 0 und max. 4 Pfeile
Gestaltung von Bildschirmsymbolen	J	Leicht zu verstehen und unterscheiden
Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung
Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Konventionen für Farbkodierung	N	Farbe dient nur der Hervorhebung der Anzeige
Anzahl der verwendeten Farbe	J	8 Farben (schwarz, weiß, rot, gelb, blau, cyan, magenta, orange) zur Auswahl, eine wird verwendet
Gesättigtes Blau	N	Blau auf gesättigtem Grün nicht zu empfehlen
Vordergrundfarben	J	Leicht zu unterscheiden, optimal - alle
Hintergrundfarben	N	Grün ist vorgegeben

B.8 TEMPOMAT-ANZEIGE

Empfehlungen	J / N	Kommentare
Platzierung der Information	J	Direkt am Objekt
Dichte der angezeigten Information	J	Max. ein Pfeil
Grafische Kodierung und Abstufungen	J	Ein Pfeil für links, einer für rechts und einer für geradeaus
Gestaltung von Bildschirmsymbolen	J	Leicht zu verstehen was mit den Pfeilen gemeint ist
Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung
Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Zugehörigkeit von Farbe zu Informationskategorien	J	Z.B. die Farbe zum Beschleunigen ist bei allen drei Pfeilarten gleich
Konventionen für Farbkodierung	J	Rot für Bremsen, Grün(hell) für Beschleunigen, Schwarz neutral
Anzahl der verwendeten Farbe	J	9 Farben (weiß, schwarz, grün, rot, gelb, blau, cyan, magenta, orange) zur Auswahl, 3 werden verwendet
Gesättigtes Blau	N	Blau auf gesättigtem Grün nicht zu empfehlen
Vordergrundfarben	J	Leicht zu unterscheiden, optimal - grün für Beschleunigen, rot für Bremsen, schwarz neutral
Hintergrundfarben	N	Grün ist vorgegeben

B.9 TACHOMETER-ANZEIGE

Empfehlungen	J / N	Kommentare

Platzierung der Information	J	Direkt unter dem Objekt
Dichte der angezeigten Information	J	Hintergrund und Ist-Geschwindigkeitsanzeige
Gestaltung von Bildschirmsymbolen	J	Leicht zu verstehen, dass es sich um einen Tachometer handelt
Richtung von Linien	J	Anhand des Hintergrunds kann man ablesen, welchem Wert die Anzeige entspricht
Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung
Angemessene Verwendung von Farbe	J	Nur vordefinierte Squeak-Farben
Konventionen für Farbkodierung	J	Grün(hell) alles gut, gelb für Vorsicht, rot für Bremsen
Anzahl der verwendeten Farbe	J	8 Farben (weiß, grün, rot, gelb, blau, cyan, magenta, orange) zur Auswahl, 3 werden verwendet
Gesättigtes Blau	N	Blau auf gesättigtem Grün nicht zu empfehlen
Vordergrundfarben	J	Die Geschwindigkeitsanzeige nur in Schwarz und leicht zu unterscheiden
Hintergrundfarben	N	Nur Farben mit hoher Sättigung

B.10 FARBÄNDERUNG OBJEKT

EMPFEHLUNGEN	J / N	KOMMENTARE
Platzierung der Information	J	Direkt auf dem Objekt
Dichte der angezeigten Information	J	Das Objekt wird nur gefärbt

Grafische Kodierung und Abstufungen	J	Das Objekt kann max. 3 Farben annehmen
Farbe als Hilfskodierung	N	Die Farbe hier ist die einzige Kodierung
Angemessene Verwendung von Farbe	J	Es werden nur 3 Farben verwendet
Konventionen für Farbkodierung	J	Grün in Ordnung, gelb Warnung, rot Bremsen
Anzahl der verwendeten Farbe	J	9 Farben (schwarz, weiß, grün, rot, gelb, blau, cyan, magenta, orange) zur Auswahl, 3 werden verwendet
Gesättigtes Blau	N	Blau auf gesättigtem Grün nicht zu empfehlen
Vordergrundfarben	J	Leicht zu unterscheiden
Hintergrundfarben	N	Grün ist vorgegeben
Kodierung von Flächen	N	Statt mit Farbe, könnten die Kästchen mit unterschiedlichen Mustern gefüllt werden

B.11 JOYSTICKANZEIGE IM OBJEKT

Empfehlungen	J / N	Kommentare
Platzierung der Information	J	Direkt im Objekt
Dichte der angezeigten Information	J	Nur die optimale Joystickposition wird im Objekt gezeigt
Farbe als Hilfskodierung	J	Zusätzlich zur der Punktanzeige
Angemessene Verwendung von Farbe	J	Es wird nur 1 eine Farbe für die Positionsanzeige und noch eine optional zum Abdecken des Objekts benutzt

Konventionen für Farbkodierung	N	Farbe dient nur der Hervorhebung der Anzeige
Anzahl der verwendeten Farbe	J	9 Farben (schwarz, weiß, rot, gelb, grün, blau, cyan, magenta, orange) zur Auswahl, max. 2 werden verwendet
Gesättigtes Blau	N	Blau ist nicht zu empfehlen
Vordergrundfarben	J	Leicht zu unterscheiden in Kombination mit der richtigen Hintergrundfarbe (z.B. gelb auf blau)
Hintergrundfarben	N	Nur Farben mit hoher Sättigung

B.12 HINDERNIS-KOLLISIONSWARNHINWEIS (DYNAMISCH)

EMPFEHLUNGEN	J / N	KOMMENTARE
Platzierung der Information	J	Optimal - Hinweis zum Beschleunigen in der oberen Bildschirmhälfte, zum Bremsen in der unteren
Dichte der angezeigten Information	J	Ein Hinweis zum Beschleunigen oder einer zum Bremsen
Gesetz der Ähnlichkeit	J	Beide Hinweise sollten entweder Textboxen sein oder beide Pfeile, aber nicht gemischt
Grafische Kodierung und Abstufungen	J	Eine Anzeige zum Beschleunigen und eine zum Bremsen
Gestaltung von Bildschirmsymbolen	J	Texte sind leicht zu lesen und Pfeilrichtungen leicht zu erkennen

Farbe als Hilfskodierung	J	Zusammen mit grafischer Kodierung; Textboxen und Pfeile sind fertige Bilddateien und werden nicht im Code gefärbt
Angemessene Verwendung von Farbe	J	Max. 2 Farben in der Anzeige
Konventionen für Farbkodierung	N	Roter Pfeil wird sowohl zum Beschleunigen als auch zum Bremsen benutzt
Gesättigtes Blau	J	Keine blaue Farbe
Chromostereopsis	J	Schwarzer Text auf weißem oder gelben Hintergrund ist in Ordnung
Hintergrundfarben	N	Grün ist vorgegeben
Kodierung durch Blinken	J	Die Aufmerksamkeit der Person auf eine mögliche Kollision mit dem dyn. Hindernis lenken
Hervorheben durch Blinken	J	Werden Textboxen als Hinweise verwendet, dann sollten diese nicht blinken

B.13 FAHRLINIE HERVORHEBEN

Empfehlungen	J / N	Kommentare
Platzierung der Information	J	Direkt auf dem Strecke
Dichte der angezeigten Information	J	Nur eine neue Fahrlinie wird gezeichnet
Kodierung mittels Linien	J	Gestrichelte und durchgezogene Linien mit unterschiedlicher Stärke
Farbe als Hilfskodierung	N	Die Farbe hier ist die einzige Kodierung
Angemessene Verwendung von Farbe	J	Die Linie wird nur in einer Farbe gefärbt

Konventionen für Farbkodierung	N	Farbe dient der Hervorhebung der Anzeige
Anzahl der verwendeten Farbe	J	6 Farben (rot, gelb, blau, cyan, magenta, orange) zur Auswahl, 1 wird verwendet
Gesättigtes Blau	J	Kann auf hellgrauen Hintergrund (Strecke) verwendet werden
Vordergrundfarben	J	Leicht zu unterscheiden
Hintergrundfarben	J	Helles Grau ist eine gute Hintergrundfarbe

B.14 TOLERANZBEREICH

Empfehlungen	J / N	Kommentare
Platzierung der Information	N	Optimal - etwas über dem Trackingobjekt, um die Hindernisse nicht zu verdecken
Dichte der angezeigten Information	N	Streckenbereich, Toleranzbereich links/rechts und der Rest; bei Gabelungen 2 Streckenbereiche und 4 Toleranzbereiche
Farbe als Hilfskodierung	N	Farbe ist hier die einzige Kodierung
Angemessene Verwendung von Farbe	J	Es werden nur 3 Farben verwendet
Konventionen für Farbkodierung	J	Grün in Ordnung (Strecke), gelb Vorsicht (Toleranzbereich) , rot Bremsen (außerhalb Toleranzbereich)
Anzahl der verwendeten Farbe	J	7 Farben (grün, rot, gelb, blau, cyan, magenta, orange) zur Auswahl, 3 werden verwendet

Gesättigtes Blau	J	Kein dunkler Hintergrund (weiß)
Vordergrundfarben	J	Leicht zu unterscheiden
Hintergrundfarben	N	Helles Weiß sollte als Hintergrund vermieden werden
Kodierung von Flächen	N	Statt mit Farbe, könnten die Bereiche mit unterschiedlichen Mustern gefüllt werden

C

KONFIGURATION DER AUTOMATIKEN

Die Tabellen in den nachfolgenden Abschnitten beschreiben die Konfigurationsparameter für die in dieser Arbeit implementierten Automatiken. Jeder Abschnitt stellt dabei die Konfigurationsmöglichkeiten von einer einzelnen Automatik vor. Der Name der Automatik entspricht nicht ihrem Klassennamen in Squeak, sondern dem Namen, mit welchem sie in AAFGT repräsentiert wird. Auf diese Weise ist es für den Anwender einfacher die Parameter zu der gewünschten Klasse zu finden. Dasselbe gilt auch für die Parameternamen, die in den nachfolgenden Tabellen in **fetter** Schrift dargestellt sind. Die Parameter zu jeder Klasse werden in derselben Reihenfolge vorgestellt, in der diese in ihrem Konfigurationsdialog von oben nach unten auftreten. In der Zeile unter dem Parameternamen ist seine Beschreibung (*kursiv*) sowie die Werte (normale Schrift), die der Parameter annehmen kann, angegeben. Der Standardwert wurde (in Klammern) gesetzt.

C.1 JOYSTICK-NAVIGATIONSANZEIGE

Joystick-Anzeige für

Für welchen MWB *ist die Anzeige vorgesehen?*
MWB 1, MWB 2, (MWB 1 und MWB 2)

Anzeigeposition: MWB **Links**
MWB **Rechts**

Die Position der Anzeige auf dem Bildschirm für die jeweilige Versuchsperson.

X: [0; Streckenbreite - Anzeigebreite] (10, 670)
Y: [0; Streckenhöhe - Anzeigehöhe] (580, 580)

MWB **eigene Ist-Position**

Soll die eigene Ist-Position angezeigt werden? Für die Konfiguration der Positionsanzeige selbst siehe nächste Tabelle.

(Ja), Nein

MWB **Partner-Ist-Position**

Soll die Ist-Position des Partners angezeigt werden? Für die Konfiguration der Positionsanzeige selbst siehe nächste Tabelle.

(Ja), Nein

MWB **Ist-Position-Summe**

Soll die Summe der Ist-Positionen angezeigt werden? Kann nur aktiviert werden, wenn die eigene und die Partner-Ist-Position ausgewählt wurden. Für die Konfiguration der Positionsanzeige selbst siehe nächste Tabelle.
Ja, (Nein)

MWB **Optimalposition**

Soll die optimale Position für den Joystick angezeigt werden? Für die Konfiguration der Positionsanzeige selbst siehe nächste Tabelle
.
Ja, (Nein)

Anzeigebreite

Bestimmt die Breite der Anzeige in Pixel.
[100; 180] (120)

Hintergrundform

Bestimmt die Form der Anzeige.
(Quadrat), Kreis, Keine

Hintergrundfarbe

Bestimmt die Hintergrundfarbe der Anzeige.
(hellgrau), blau, rot, gelb, transparent, cyan, orange, grün, weiß, magenta, schwarz

Rahmenfarbe

Bestimmt die Rahmenfarbe der Anzeige.
blau, rot, gelb, cyan, orange, grün, weiß, magenta, (schwarz)

Die Parameter aus der Tabelle oben, die bestimmen ob die eigene Ist-Position, die Ist-Position des Partners, deren Summe und die optimale Joystickposition angezeigt werden soll, legen nur fest, was angezeigt werden soll aber nicht wie es auszusehen hat. Die Tabelle unten enthält genau diese Parameter. Sie sind für alle Positionsanzeigen gleich.

Anzeigetyp

Bestimmt den Typ der Positionsanzeige.
Kreis ●
Linie —
Linie mit Kreis —●
Linie mit Pfeil —▶

Linientyp

Bestimmt die Art der Linie. Kann nur dann nicht verwendet werden, wenn als Anzeigetyp "Kreis" ausgewählt wurde.

durchgezogen ——
gestrichelt - - -

Farbe

Bestimmt die Farbe der Positionsanzeige.
blau, rot, gelb, cyan, orange, grün, weiß, magenta, schwarz

C.2 STEUERUNGSREGULIERUNG

Anzeige der Regulierung für

Für welchen MWB ist die Anzeige vorgesehen?
MWB 1, MWB 2, (MWB 1 und MWB 2)

Anzeigeposition: Pfeil Links
Pfeil Rechts

Die Position der Anzeige auf dem Bildschirm für die jeweilige Versuchsperson.

X: [0; Streckenbreite - Anzeigebreite] (15, 665)
Y: [0; Streckenhöhe - Anzeigehöhe] (585, 585)

Pfeilfarbe

Bestimmt die Farbe des angezeigten Pfeils.
blau, gelb, cyan, orange, weiß, magenta, (schwarz)

C.3 GESCHWINDIGKEITSREGULIERUNG (PFEIL)

Abweichungstoleranz

Bestimmt die Abweichung der vertikalen Geschwindigkeit, mit der jeder der beiden Personen das Objekt vorwärts bewegt, von der Idealgeschwindigkeit des Objekts für den aktuellen Streckenabschnitt. Ist die Abweichung für eine der Personen größer als die angegebene, dann wird seine Anzeige aktiviert.

[0; 10] (0)

Anzeige der Abweichung für

Für welchen MWB ist die Anzeige vorgesehen?
MWB 1, MWB 2, (MWB 1 und MWB 2)

Anzeigeposition: Pfeil Links
Pfeil Rechts

Die Position der Anzeige auf dem Bildschirm für die jeweilige Versuchsperson. Da die beiden Pfeile immer auf einer Höhe sein müssen, kann für die Y-Achse nur einen Wert eingegeben werden. Auf der X-Achse können die Pfeile auf beiden Seiten nur bis zur Mitte der Strecke bewegt werden.

X1: [0; Streckenbreite/2 - Pfeilbreite] (60)
X2: [Streckenbreite/2; Streckenbreite - Pfeilbreite] (690)
Y: [0; Streckenhöhe - Hintergrundhöhe] (640)

Pfeilfarbe

Bestimmt die Farbe des angezeigten Pfeils.
blau, gelb, cyan, orange, rot, grün, weiß, magenta, (schwarz)

Hintergrundtransparenz aktiv

Aktiviert oder deaktiviert die Hintergrundtransparenz.
(Ja) - [10; 100] (50)
Nein

C.4 GESCHWINDIGKEITSREGULIERUNG (KÄSTCHEN)

Anzeige der Regulierung für

Soll nur der Teil zum Beschleunigen oder nur der Teil zum Abbremsen der Anzeige oder beides angezeigt werden?

Beschleunigen, Abbremsen, (Beschleunigen und Abbremsen)

Anzeigeposition: Beschleunigen
Abbremsen

Die Position der beiden Teile der Anzeige auf dem Bildschirm.

X: [0; Streckenbreite - Teilbreite] (160, 455)
Y: [0; Streckenhöhe - Teilhöhe] (160, 40)

Farben

Bestimmt die Farbe der Kästchen und die Farbe ihres Rahmens. Die Kästchen sind von 1 bis 4 durchnummeriert. Die Nummerierung entspricht der Größe der Kästchen. Der Rahmen ist nur bei leeren Kästchen sichtbar.

orange, weiß, magenta, rot, grün, cyan, blau, schwarz, gelb
Kästchen 1 (schwarz)
Kästchen 2 (gelb)
Kästchen 3 (orange)
Kästchen 4 (rot)
Rahmenfarbe (schwarz)

Abstand

Bestimmt den Abstand zwischen den Kästchen in Pixel.
[1; 30] (10)

Transparenz aktiv

Aktiviert oder deaktiviert die Anzeigentransparenz.
Ja - [10; 90] (50)
(Nein)

C.5 STEUERUNGSVORGABENANZEIGE

Anzeige der Vorgabe für Steuerung

Soll nur der horizontale oder nur der vertikale Schieber oder beide angezeigt werden?

Horizontal, Vertikal, (Horizontal und vertikal)

Position: Schieber horizontal
Schieber vertikal

Die Position der beiden Schieber auf dem Bildschirm.

X: [0; Streckenbreite - Schieberbreite] (225, 580)
Y: [0; Streckenhöhe - Schieberhöhe] (738, 495)

Farben

Bestimmt die Farben für den Schieberbereich und -wert.

blau, cyan, orange, grün, rot, gelb, magenta
Schieberbereich (rot)
Schieberwert (gelb)

Transparenz aktiv

Aktiviert oder deaktiviert die Anzeigentransparenz.
Ja - [10; 90] (50)
(Nein)

C.6 PFEILE AM OBJEKT

MWB **Links Ist-Position**
Soll die Ist-Position für die linke Versuchsperson angezeigt werden? (Ja), Nein

MWB **Rechts Ist-Position**
Soll die Ist-Position für die rechte Versuchsperson angezeigt werden? (Ja), Nein

MWB **Ist-Position-Summe**
Soll die Summe der Ist-Positionen angezeigt werden? Wählbar nur wenn die Anzeige der Ist-Positionen beider Versuchspersonen aktiviert ist. (Ja), Nein

MWB **Optimalposition**
Soll die optimale Position (gemeinsam) für beide Personen angezeigt werden? (Ja), Nein

Neben der Auswahl für an oder aus für die Positionsanzeigen aus der obigen Tabelle, bietet der Konfigurationsdialog auch Parameter zum Einstellen der Positionsanzeige selbst. Diese sind in der Tabelle unten aufgelistet.

Linientyp
Bestimmt die Art der Linie (der Pfeilteil ohne die Spitze). durchgezogen —— gestrichelt - - -

Farbe
Bestimmt die Farbe der Positionsanzeige (des Pfeils). blau, rot, gelb, cyan, orange, grün, weiß, magenta, schwarz

Anzeige (nur für Optimalposition)
Bestimmt die Art der Anzeige. Bei "Geschwindigkeit und Richtung" ändert der Pfeil außer seine Richtung auch seine Länge. Dies deutet auf eine Änderung der Geschwindigkeit hin. (Geschwindigkeit und Richtung), Nur Richtung

C.7 STEUERUNGSREGULIERUNG AM OBJEKT

Hütchenfarbe

Bestimmt die Farbe der Hütchen (der Pfeile).

blau, rot, gelb, cyan, orange, weiß, magenta, (schwarz)

C.8 TEMPOMAT-ANZEIGE

Pfeilfarben

Bestimmt die Farbe der Tempomat-Pfeile.

blau, rot, gelb, grün, cyan, orange, weiß, magenta, schwarz
Bei optimaler Geschwindigkeit (schwarz)
Zum Gasgeben (grün)
Zum Bremsen (rot)

C.9 TACHOMETER-ANZEIGE

Zulässige Abweichung

Bei einer Abweichung der Objektgeschwindigkeit von der für den Streckenabschnitt optimalen Geschwindigkeit wird der Tachometer-Hintergrund in 3 verschiedene Farben gefärbt. Dieser Parameter bestimmt die maximale Abweichung für jede Farbe.

Optimale Geschwindigkeit: [0; 10] (10)
Warnstufe 1: [0; 20] (20)
Warnstufe 2: > Wert für Warnstufe 2

Tachometerfarbe

Bestimmt die Farbe des Tachometer-Hintergrunds. Die Geschwindigkeitsanzeige auf dem Tachometer bleibt immer schwarz.

gelb, orange, rot, magenta, grün, weiß, blau, cyan
Optimale Geschwindigkeit: (grün)
Warnstufe 1: (gelb)
Warnstufe 2: (rot)

C.10 FARBÄNDERUNG OBJEKT

Sensoren außerhalb der Strecke

Das Objekt wird in Abhängigkeit der Anzahl der Sensoren, die die grüne Farbe außerhalb der Strecke sehen, in 3 verschiedene Farben gefärbt. Rund ums Objekt sind 8 Farbsensoren platziert. Dieser Parameter bestimmt die maximale Sensoranzahl für jede Farbe.

[0; 8]
Bereich I: (0)
Bereich II: (3)
Bereich III: > Bereich II

Farbe

Bestimmt die Objektfarbe für jeden Bereich.

gelb, orange, rot, magenta, grün, weiß, blau, cyan, schwarz
Bereich I: (grün)
Bereich II: (gelb)
Bereich III: (rot)

C.11 JOYSTICKANZEIGE IM OBJEKT

Farbe

Bestimmt die Farbe des Punktes, der die gemeinsame optimale Joystickposition visualisiert.

gelb, orange, (rot), magenta, grün, weiß, blau, cyan, schwarz

Objektgeschwindigkeit verstecken

Soll die in SAM eingebaute Ist-Geschwindigkeitsanzeige im Objekt versteckt werden? Wenn ja, welche Farbe soll das Objekt annehmen.

Ja - gelb, orange, rot, magenta, grün, weiß, blau, cyan, (schwarz)
(Nein)

C.12 HINDERNIS-KOLLISIONSWARNHINWEIS (DYNAMISCH)

Warnung durch Hinweis für

Sollen die Versuchspersonen durch einen Hinweis zum Beschleunigen oder zum Abbremsen oder beides vor einer möglichen Kollision mit dem dynamischen Hindernis gewarnt werden?

(Beschleunigen oder Abbremsen), Nur Beschleunigen, Nur Abbremsen

Bilder und Positionen: Beschleunigen Abbremsen

Die Position der Hinweise auf dem Bildschirm und die Auswahl der Bilder die als Hinweis angezeigt werden. Nach einem Klick im Bildauswahlbereich bekommt der Anwender eine Liste mit allen im System hierfür vorgesehenen Bildern.

X: [0; Streckenbreite - Hinweisbreite] (660, 660)
Y: [0; Streckenhöhe - Hinweishöhe] (0, 728)

Transparenz aktiv

Aktiviert oder deaktiviert die Anzeigentransparenz.
Ja - [10; 90] (50)
(Nein)

Blinken aktiv

Aktiviert oder deaktiviert das Blinken der Anzeigen.
Ja, (Nein)

Wenn Blinken aktiv ist, dann kann man zusätzlich die Parameter aus der folgenden Tabelle einstellen.

Einblendedauer, Ausblendedauer

Bestimmt die Dauer (in Millisekunden), für die die Anzeige eingeblendet und ausgeblendet wird.

[100, 5000]
Einblendedauer: (200)
Ausblendedauer: (200)

C.13 FAHRLINIE HERVORHEBEN

Linientyp

Bestimmt, ob die Mittel- oder die Ideallinie angezeigt wird.

(Mittellinie), Ideallinie

Linienart

Bestimmt die Art der gezeichneten Linie. Die Ideallinie kann nicht gestrichelt gezeichnet werden.

(durchgezogen) ——
gestrichelt - - -

Linienfarbe

Bestimmt die Farbe für die Linie.

blau, cyan, magenta, orange, rot, (gelb)

Linienstärke

Bestimmt die Dicke (Stärke) der Linie.

[1; 6] (4)

Gabelungen

In welchem Zweig, im Falle einer Gabelung, soll die Linie gezeichnet werden? Es existieren nur eckige und runde Gabelungen.

Besserer/Optimaler Zweig, Dünner Zweig, Breiter Zweig, Kurzer Zweig, Langer Zweig, Schneller Zweig, Rechter Zweig, Einfacher Zweig, Schwieriger Zweig, Zufälliger Zweig

Runde Gabelung: (Besserer/Optimaler Zweig)
Eckige Gabelung: (Besserer/Optimaler Zweig)

C.14 TOLERANZBEREICH

Toleranzbereichbreite

Bestimmt die Breite des Toleranzbereichs (in Pixel) links und rechts von der Strecke.
[10, 120] (60)

Bereichsfarben

Bestimmt die Farben der 3 Bereiche in der Anzeige.

cyan, blau, orange, grün, rot, gelb, magenta
Streckenbereich: (grün)
Toleranzbereich: (gelb)
Restbereich: (rot)

Position

Bestimmt die vertikale Position der Anzeige.

Y: [0; Streckenhöhe - Anzeigehöhe] (0)

D

INHALT DER DVD

Zusätzlich zu dieser Diplomarbeit in Papierform wurde bei der Abgabe auch eine DVD eingereicht. Auf dieser befinden sich:

1. Die Datei “Diplomarbeit_AydanSeid.pdf”
   Die elektronische Fassung dieser Arbeit.

2. Das Verzeichnis “Squeak”
   Quellcode zusammen mit den in dieser Arbeit implementierten Klassen und ein lauffähiges Squeak-Image

   - Die Liste der implementierten Klassen ist in Anhang A zu finden.
   - Das Image startet mit geöffnetem AAFGT, in dem alle in dieser Arbeit implementierten Agenten zur Verwendung bereits ausgewählt wurden.

SELBSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Diplomarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 17. Dezember 2012 .....