Ein Ereignis-System für das ATEO Automation Framework sowie die Implementierung und Testung von auditiven und visuellen Hinweisen

Diplomarbeit

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

eingereicht von: Nikolai Kosjar
geb. am 14.05.1986 in Rivne (Ukraine)

am: 22. März 2012

Gutachter: Herr Prof. Dr. Klaus Bothe
Herr Prof. Dr. Hartmut Wandke

Betreuerin: Dipl.-Psych. Saskia Kain

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

AKRONYME

ATEO Arbeitsteilung Entwickler-Operateur

prometei Prospektive Gestaltung von Mensch-Technik-Interaktion

AAF ATEO Automation Framework

SAM Socially Augmented Microworld

MWB Mikroweltbewohner

GUI Graphical User Interface

MVC Model View Controller

1 EINLEITUNG

In diesem Abschnitt wird das Projekt vorgestellt, in welches die Diplomarbeit eingebettet ist. Darauf aufbauend wird die Motivation erklärt und das Ziel dieser Arbeit definiert. Schließlich wird die Gliederung der nachfolgenden Kapitel vorgestellt.

1.1 DAS ATEO-PROJEKT

Das Projekt Arbeitsteilung Entwickler-Operateur (ATEO) ist ein Beitrag zum interdisziplinären Graduiertenkolleg Prospektive Gestaltung von Mensch-Technik-Interaktion (prometei), welches am Zentrum für Mensch-Maschine-Systeme der Technischen Universität Berlin organisiert ist. Wissenschaftler des Fraunhofer Instituts für Produktionsanlagen und Konstruktionstechnik unterstützen das Graduiertenkolleg, ebenso wie der Lehrstuhl für Ingenieurpsychologie und Kognitive Ergonomie der Humboldt-Universität zu Berlin mit dem ATEO-Projekt.

Im ATEO-Projekt wird die Arbeitsteilung zwischen zwei verschiedenen Gruppen experimentell variiert und empirisch untersucht. Der ersten Gruppe gehören Entwickler bzw. Designer an, die Systeme antizipieren und planen. Der zweiten Gruppe gehören Operateure bzw. Anwender an, welche die implementierten Systeme nutzen. In den Experimenten wird die Antizipation von Ereignissen durch den Einsatz der durch die Entwickler konzipierten Automatiken mit der Reaktion auf Ereignisse von Operateuren verglichen.

Abbildung 1.1 zeigt die im ATEO-Projekt verwendeten Software-Komponenten. Diese sind im Folgenden kurz beschrieben.

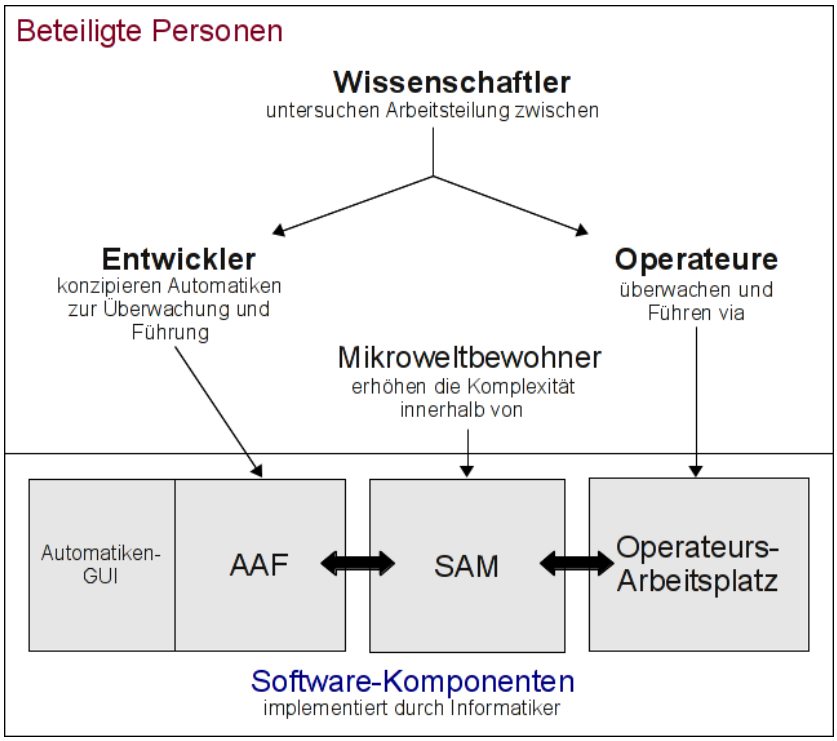


Abbildung 1.1: Personen und Software-Komponenten im ATEO-Projekt

SAM Der zentrale Bestandteil des ATEO-Projekts wird von Bothe u. a. (2009) beschrieben. Es handelt sich um die Mikrowelt mit dem Namen SAM. Die Mikroweltbewohner (MWB) sind lebendige Komponenten und bringen die notwendige Komplexität ins System. Abschnitt 1.1.1 stellt SAM näher vor.

AAF Das ATEO Automation Framework (AAF) von Hasselmann (2012) bildet die technische Grundlage, damit die von den Entwicklern spezifizierten Systeme (Automatiken) Einfluss auf SAM nehmen können. Abschnitt 1.1.2 beschreibt das Framework genauer.

AUTOMATIKEN-GUI Die von Fuhrmann (2010) entwickelte Benutzungsschnittstelle baut auf der AAF-Komponente auf und erlaubt es den Forschern, die von den Entwicklern spezifizierten Automatiken unkompliziert zu erstellen und zu konfigurieren. Abschnitt 1.1.3 stellt das Graphical User Interface (GUI) vor.

OPERATEURSARBEITSPLATZ Der Operateursarbeitsplatz ermöglicht die Überwachung eines SAM-Versuchsablaufs und den Eingriff in diesen. Schwarz (2009) hat ihn in den Grundzügen entwickelt und Leonhard (2012) kümmert sich aktuell um die Weiterentwicklung. Der Operateursarbeitsplatz ist für die vorliegende Arbeit kaum relevant und wird hiermit nur der Vollständigkeit halber erwähnt.

1.1.1 *Socially Augmented Microworld (SAM)*

Bei SAM handelt es sich um eine Mikrowelt, in der die MWB die Aufgabe haben, ein rundes Objekt entlang einer vorgegebenen Strecke zu navigieren. Dabei haben beide MWB Einfluss auf die Steuerung des Objekts.

Die Aufgabe besteht darin, das Objekt möglichst schnell und möglichst genau entlang der Strecke zu navigieren. Ein MWB soll die Priorität auf die Genauigkeit des Fahrens legen, der andere auf die Schnelligkeit.

Die Tracking-Aufgabe wird zusätzlich erschwert, indem durch Gabelungen und Hindernisse auf der Strecke Konfliktsituationen hervorgerufen werden. Abbildung 1.2 vermittelt einen Eindruck von SAM.

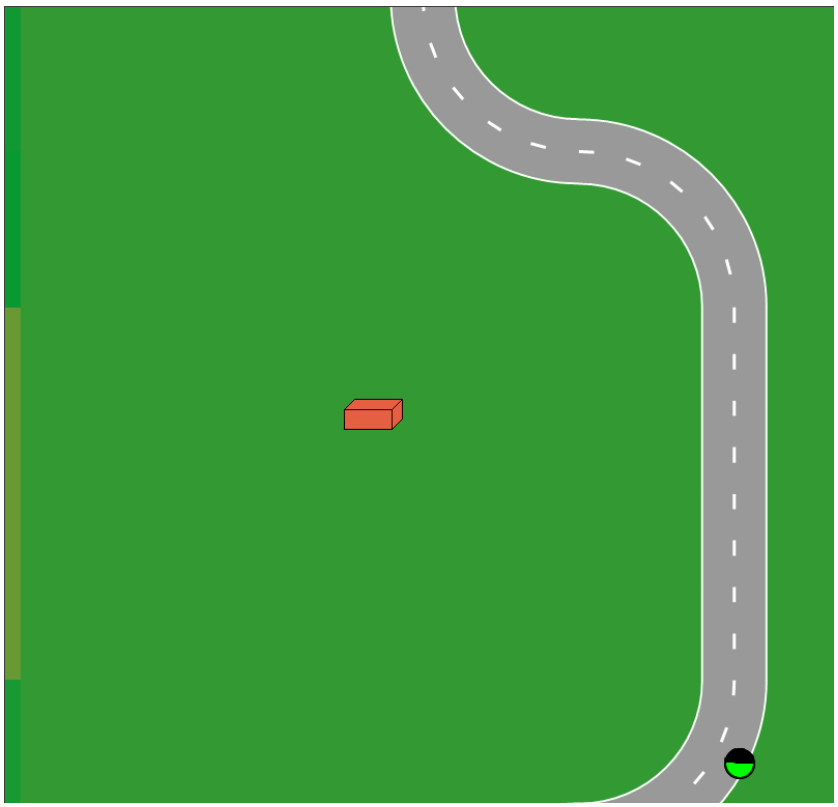


Abbildung 1.2: SAM - ein sich näherndes Hindernis verursacht eine Konfliktsituation

1.1.2 *ATEO Automation Framework (AAF)*

Eine Automatik dient der Prozessüberwachung und -führung von SAM durch indirekte Eingriffe über Informationen an die MWB oder direkte Eingriffe in die Steuerung des Objekts. So kann eine Automatik zum Beispiel verhindern, dass das Objekt sich abseits der Fahrbahn befindet. Denkbar ist auch, dass auftretende

Konfliktsituationen durch die Anzeige von Hinweisen entschärft werden.

Um Automatiken in SAM integrieren zu können, entwickelt Hasselmann (2012) in seiner Diplomarbeit die AAF-Komponente. Automatiken sind Graphen und bestehen aus Knoten, die entweder Funktionen oder andere Automatiken sind. Ein Graph enthält zwei besondere Knoten - eine Wurzel und eine Senke. Jeder Knoten, Wurzel und Senke ausgenommen, hat genau einen Vorgänger und genau einen Nachfolger. Über die Wurzel kommt ein Teilzustand von der zentralen SAM-Datenstruktur SAMModelData in die Graphen - der SAMState. Die direkten Kinderknoten der Wurzel erhalten jeweils eine Kopie des SAMState und können ihn manipulieren und diesen weiter an ihre direkten Kinderknoten weitergeben, welche weitere Änderungen vornehmen können[1]. Dieser Vorgang setzt sich fort, bis die veränderten und weitergereichten SAMStates schließlich in die Senke „münden“. Die Senke erzeugt einen SAMState, welcher in SAM eingespeist wird und dessen Werte in SAMModelData übernommen werden - auf diese Weise haben die Funktionen Einfluss auf die Verarbeitung in SAM. Eine Besonderheit des AAF ist es, dass jeder Knoten wiederum ein Graph, d.h. eine andere Automatik, sein kann. Da jeder Graph genau eine Wurzel und eine Senke hat und jeder Knoten in einem Graphen genau einen Vor- und genau einen Nachfolger hat, kann ein beliebiger Knoten durch einen Graphen ersetzt werden. Auf diese Art und Weise können selbst komplexe Automatiken entwickelt werden.

Funktionen und Automatiken können demnach zu neuen Automatiken zusammengesetzt werden. Um das auch ohne Programmierkenntnisse durchführen zu können, wurde das Automatiken-GUI entwickelt.

1.1.3 *Automatiken-GUI*

Im Rahmen ihrer Diplomarbeit hat Fuhrmann (2010) eine grafische Benutzungsschnittstelle entwickelt, mit der Forscher für ihre Experimente komplexe Automatiken erstellen und konfigurieren können. Das GUI wurde von Kosjar (2011) hinsichtlich der Gebrauchstauglichkeit überarbeitet und sieht aktuell wie in Abbildung 1.3 gezeigt aus.

Im Einzelnen können mit dieser Anwendung Automatiken erstellt, gespeichert, geladen und bearbeitet werden. Zum Bearbeiten gehört auch das Verändern von Parametern einzelner Funktionen sowie die Aktivierung bzw. Deaktivierung von Funktionen oder Automatiken für einzelne Streckenabschnitte. Das Erstellen von Automatiken wird durchgeführt, indem Funktio-

1 Funktionen können demnach auch Änderungen ihrer Vorgänger verändern.

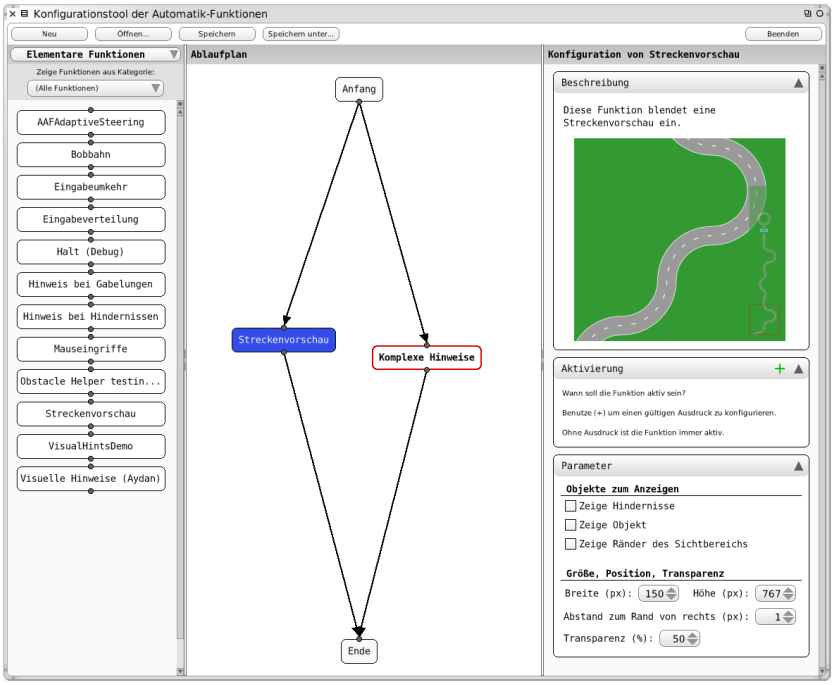


Abbildung 1.3: Das Automatiken-GUI zum Erstellen und Konfigurieren von Automatiken

nen und bisher erstellte Automatiken als Bausteine in einem Ablaufplan zusammengesetzt werden.

1.2 MOTIVATION UND ZIEL DIESER ARBEIT

Um den Beitrag der Entwickler in die Untersuchungen einzubringen, sollen die Forscher die von den Entwicklern konzipierten Automatiken erstellen und konfigurieren können. Die Konzepte für die Automatiken wurden von professionellen Entwicklern erarbeitet. Kain (2012) hat insgesamt 30 Teams, bestehend aus drei Personen, die Aufgabe gestellt Automatiken zu entwickeln, die zur Überwachung und im Notfall bei Prozessabweichung zur Führung von SAM agieren sollen.

Das Ergebnis der Entwicklung ist die Beschreibung von Konzepten bestehend aus einzelnen Funktionen. Diese sind auf Papier in „Konzeptbögen“ dokumentiert und anschließend von Kain digitalisiert worden.

Aktuell werden die Konzepte von einer Gruppe angehender Informatiker unter der Leitung von Kain implementiert. Diese Arbeit stellt dazu einen Beitrag dar.

Das Ziel dieser Arbeit ist die Implementierung von allen sogenannten weichen Eingriffen, die als Funktionen in den Konzeptbögen beschrieben sind und nichts mit Soll- und Ist-Anzeigen[2] zu tun haben. Bei weichen Eingriffen handelt es sich um auditi-

2 Diese speziellen Hinweise werden aktuell von Aydan Seid implementiert.

ve und visuelle Hinweise mit Vorschlagscharakter. Jedes Team erachtet Hinweise als wichtiges Hilfsmittel um die MWB bei ihrer Aufgabe zu unterstützen. Entsprechend vielfältig fallen die gewünschten Hinweise aus.

Diese Arbeit umfasst also die Analyse aller relevanten Hinweis-Funktionen, um diese anschließend sinnvoll zu implementieren. Zur Implementierung gehört auch die Integration der Funktionen in das in Abschnitt 1.1.3 erwähnte Automatiken-GUI, damit die Forscher diese für ihre Untersuchungen anpassen können. Aufgrund der Vielfalt der Hinweise gibt es eine relativ große Anzahl an Parametern zum Einstellen. Für die Integration in das GUI sollen die Grundsätze der Software-Ergonomie berücksichtigt werden. Durch automatische und manuelle Tests soll die Funktionalität der Hinweise sichergestellt werden. Wie die anderen Software-Komponenten im ATEO-Projekt, werden die Hinweise in der Entwicklungsumgebung Squeak umgesetzt. Bei dieser kommt die Programmiersprache Smalltalk zum Einsatz.

Im Wesentlichen basiert diese Arbeit auf der AAF-Komponente von Hasselmann (2012), der Automatiken-GUI von Fuhrmann (2010) und natürlich der Arbeit von Kain (2012).

1.3 GLIEDERUNG DIESER ARBEIT

Diese Arbeit ist in mehrere Kapitel gegliedert.

In Kapitel 2 werden dem Leser allgemeine Voraussetzungen zum Verständnis der nachfolgenden Kapitel verständlich gemacht. In Kapitel 3 wird auf Prozess-Modelle der Software-Entwicklung sowie die Gestaltung von und Interaktion mit Benutzungsschnittstellen eingegangen. In Kapitel 4 werden die Ergebnisse der Analyse der sogenannten Konzeptbögen beschrieben. Aufbauend darauf werden in Kapitel 5 die zu implementierenden Anforderungen festgehalten. Einblicke in die Implementierung enthält der Leser in Kapitel 6. Die Validierung der Implementierung wird in Kapitel 7 beschrieben. Schließlich wird in Kapitel 8 die Arbeit zusammengefasst und kritisch reflektiert.

Diese Kapitel werden durch mehrere Anhänge vervollständigt. In Anhang A wird aufgezeigt, welche Funktionen aus den Konzeptbögen implementiert werden konnten. Anhang B und C sind Anleitungen, die während dieser Arbeit entstanden sind (Anhang B) oder ergänzt wurden (Anhang C). Im letzten Anhang, Anhang D, ist kurz beschrieben, was auf der beigelegten DVD zu finden ist.

2 VORAUSSETZUNGEN

In diesem Kapitel werden wichtige Voraussetzungen geklärt, um die nachfolgenden Kapitel zu verstehen.

Neben der Programmiersprache Smalltalk (Abschnitt 2.1), der Entwicklungsumgebung Squeak (Abschnitt 2.2) und dessen Framework zum Gestalten von Benutzungsschnittstellen (Abschnitt 2.3), werden auch die zwei grundlegenden Klassen des Smalltalk Grafik-Kernels vorgestellt (Abschnitt 2.4). Außerdem wird auf Einzelheiten von SAM eingegangen (Abschnitt 2.5) sowie kurz erklärt, wie eine neue Funktion im AAF realisiert werden kann (Abschnitt 2.6). Schließlich wird vorgestellt, welcher IST-Zustand von SAM bezüglich der auditiven und visuellen Hinweise (Abschnitt 2.7) vorgefunden wurde.

2.1 SMALLTALK

„Smalltalk“ bezeichnet eine Programmiersprache, die zugehörige Entwicklungsumgebung sowie die virtuelle Maschine, mit deren Hilfe die Smalltalk-Programme ausgeführt werden. Je nach Literatur wird daher auch von „Smalltalk-Systemen“ geredet. Durch verschiedene Implementierungen von Smalltalk-Systemen gibt es inzwischen auch nicht eine Smalltalk-Programmiersprache, viel mehr hat sich eine Gruppe von Sprachdialekten entwickelt. (Gray u. Mohamed 1990, S. 1f)

Das erste Smalltalk-System wurde in den 70er Jahren im Palo Alto Research Center (PARC) von Xerox entwickelt. Nach fünf mehrjährigen Entwicklungszyklen wurde das System 1981 unter dem Namen Smalltalk-80 als erste kommerzielle Version von Xerox veröffentlicht. Adele Goldberg, Daniel Ingalls und Alan Key waren die wichtigsten Entwickler.[1]

Die Smalltalk-Programmiersprache basiert nur auf wenigen Konzepten, sodass die Sprache relativ schnell gelernt und verstanden werden kann. Es ist eine dynamisch typisierte[2] und rein objektorientierte Sprache. Die Aussage „Alles ist ein Objekt“ gilt demnach z. B. auch für Integer und Zeichen. Mit Smalltalk wurde Pionier-Arbeit an grafischen Benutzungsschnittstellen geleistet. Sehr populär wurde u. a. das Konzept Model View Controller (MVC), welches von Trygve Reenskaug für eine klare

1 vgl. Stritzinger 1997, S. 21f. und Gray u. Mohamed 1990, S. 1
2 Der Typ einer Variablen wird erst bei der Ausführung bestimmt.

Trennung von eigentlicher Funktionalität („Business-Logic“) und Benutzungsschnittstelle eines Programms entwickelt wurde.[3]

Zu den bekannten kommerziellen Smalltalk-Systemen zählen „VisualAge Smalltalk“ von IBM und „Visual Works“ von Cincom. In der Open-Source-Welt ist „GNU Smalltalk“ und „Squeak“ verbreitet. Letzteres findet auch im ATEO-Projekt Verwendung.

2.2 SQUEAK

Squeak ist eine Open-Source-Implementierung eines Smalltalk-Systems. Es entstand bei Apple Inc. durch eine Forschungsgruppe, welche Prototypen für Lernsoftware und Experimente mit Benutzungsschnittstellen entwickelte. Daneben hatte die Forschungsgruppe auch zum Ziel das Dynabook-Konzept von Alan Kay umzusetzen. Dieses Konzept beschreibt einen tragbaren Computer, der speziell für Kinder aller Altersklassen gedacht ist. Neben Alan Kay selbst zählen Dan Ingalls, Ted Kaehler und Scott Wallace zu den wichtigsten Entwicklern.

Dem Dynabook-Konzept folgend bietet Squeak die Möglichkeit jeden Teil des Smalltalk-Systems einzusehen und zu studieren, auch die virtuelle Maschine, welche als Besonderheit ebenfalls in Smalltalk implementiert ist.

Multimedia wird von Squeak besonders unterstützt. Durch Plugins für die virtuelle Maschine stehen Features wie z. B. 2D mit Kantenglättung, beschleunigtes 3D, Echtzeit Sound- und Musiksynthese, Unterstützung für MPEG2-Videos, und viele andere mehr zur Verfügung. Insgesamt existieren über 750 einfach nachinstallierbare Pakete[4].

Die Benutzeroberfläche von Squeak heißt „Morphic“. Auf diese wird im nächsten Abschnitt eingegangen. Es lassen sich aber weiterhin Benutzeroberflächen erstellen, welche auf dem traditionellen MVC-Konzept basieren. Tatsächlich kann der Benutzer beides kombinieren.

Für eine vollständige Liste von Features sei auf die offizielle Squeak-Webseite[5], insbesondere die „About“-Seiten, verwiesen.

2.3 DAS MORPHIC-FRAMEWORK VON SQUEAK

Das Morphic-Framework von Squeak gestattet es auf sehr einfache Art und Weise interaktive und lebendige Benutzeroberflächen zu erstellen. Es verfügt über vernünftige Voreinstellungen und nimmt dem Entwickler viel Arbeit ab, wenn es um Drag & Drop, Animationen, automatisches Layout von Morphen und Ereignisverarbeitung geht.

3 vgl. Goldberg u. Robson 1989, S. viii und Gray u. Mohamed 1990, S. 1
4 vgl. http://map.squeak.org/ (abgerufen am 18. März 2012, 17:57 Uhr)
5 vgl. http://www.squeak.org/ (abgerufen am 18. März 2012, 18:00 Uhr)

Abbildung 2.1: Links: Ein Morph in seinen Standardeinstellungen; Rechts: Morph mit veränderter Hintergrundfarbe und angezeigtem Halo.

Das zentrale Element im Morphic-Framework ist der „Morph“ (griechisch für Gestalt bzw. Form). Wird ein neuer Morph erstellt und angezeigt, z. B. mit `Morph new openInWorld.`, dann erscheint ein blaues Rechteck, wie in der Abbildung 2.1 links zu sehen ist. Dieser Morph ist jedoch kein flüchtiges Rechteck, welches verschwindet, sobald ein Fenster drüber und wieder weggeschoben wird. Es ist ein interaktives Objekt. Per Drag & Drop kann z. B. seine Position verändert werden. Weitere Einstellungen zur Manipulation des Morphs erhält der Benutzer, wenn er den zugehörigen „Halo-Morph“ anzeigen lässt - z. B. durch Drücken der mittleren Maustaste auf den Morph. Es erscheinen dann um den Morph verschiedene, anklickbare Buttons mit Icons (in Abbildung 2.1 rechts sichtbar). Verweilt der Benutzer mit der Maus über einen Button, so erscheint ein Tooltip, der die zugehörige Manipulation erklärt. Zu den Manipulationen zählen u. a. Rotation, Kollabieren, Entfernen, Menü aufrufen, Verschieben, Duplizieren, Debuggen, Farbe ändern, Vergrößern. Der grüne Morph rechts in der Abbildung 2.1 ist durch Duplikation und Farbänderung entstanden. Weitere Manipulationsmöglichkeiten erschließen sich dem Benutzer, wenn er das Menü aufruft (das zugehörige Halo-Icon ist rot und zeigt ein kleines Menü). Ein Entwickler, der einen Morph erstellt, kann dem Menü individuelle Einträge hinzufügen.

Um eigene Morphe oder Bedienelemente zu erstellen, kann der Benutzer wie eben beschrieben vorgehen: Einen Morph erzeugen und ihn solange über den Halo und das Menü verändern, bis er den Wünschen entspricht. Morphe können auch interaktiv verschachtelt bzw. zusammengesetzt werden. Auf diese Art und Weise lassen sich Morphe ohne Programmierkenntnisse erstellen. Das ist z. B. zum Erstellen von GUI-Prototypen sehr nützlich. Für weiterführende Möglichkeiten, wie z. B. Animationen oder dem Verhalten bei bestimmten Mausinteraktionen oder Tastatureingaben, muss Quelltext geschrieben werden. Dazu wird eine neue Klasse erstellt, die von der Klasse `Morph` erbt. Die Methoden, die hinzugefügt oder überschrieben werden müssen, sind übli-

cherweise sehr klein, sodass mit relativ wenig Quellcode viel im Morphic-Framework erreicht werden kann. Auch lässt sich durch die Definition einer eigenen Klasse das Aussehen des Morphs sehr individuell gestalten: In der Methode #drawOn: lassen sich auf einem FormCanvas die üblichen Zeichenoperationen (Zeichnen von u. a. Polygonen, Kurven, Ellipsen, Linien, Bildern und Text) durchführen.

Die Oberfläche von Squeak ist auch in Morphic implementiert. Der Hintergrund (Klasse PasteUpMorph), das Fenster-System (Klasse SystemWindow) und der Mauszeiger (Klasse HandMorph) sind Morphe. Viele Tools von Squeak, z. B. der Browser, beruhen aber teilweise noch auf dem MVC-Konzept. Diese Tools zeigen, dass sich beide Frameworks gemeinsam verwenden lassen[6]. Im MVC-Kontext betrachtet, übernimmt ein Morph die View- und Controller-Rolle. Der Morph kann zugleich aber auch das Model sein, muss es aber nicht, denn durch das Observer-Pattern[7] kann über #changed: und #update: weiterhin eine klare Trennung der Darstellung und Interaktion (View und Controller) vom Model gewährleistet werden.

Das Morphic-Framework von Squeak wird von Maloney (2000), Maloney (2011) genauer beschrieben.

2.3.1 *Owner und Submorphe*

Morphe werden in einer Baumstruktur organisiert. Ein Morph hat demnach einen Eltern-Knoten, *Owner* genannt, sowie potenzielle Kinderknoten, *Submorphe* genannt. Der Wurzelknoten in dieser Baumstruktur wird *World* genannt und bezeichnet auch das gleichnamige Objekt. Wird ein neuer Morph erstellt und angezeigt, z. B. mit Morph new openInWorld, dann ist dieser automatisch ein *Submorph* von *World*.

Die Eltern-Kind-Beziehungen können auch explizit durch owner addMorphBack: submorph definiert werden, wie das folgende Listing 2.1 zeigt.

Listing 2.1: Definition von Owner und Submorph

```
newParentMorphWithColor := [ :c |
  (Morph new color: c)
    changeTableLayout; layoutInset: 10; cellInset: 5;
    hResizing: #shrinkWrap; vResizing: #shrinkWrap;
    listDirection: #leftToRight.
  ].

subparent := newParentMorphWithColor value: (Color red).
```

6 Dazu wurden die typischen View- und Controller-Klassen in entsprechende Morphe eingebettet.

7 Ein Entwurfsmuster der objekt-orientierten Programmierung, durch welches Objekte andere abhängige Objekte über Änderungen benachrichtigen können.

```
subparent addMorphBack: (Morph new color: Color blue).
subparent addMorphBack: (Morph new color: Color green).

parent := newParentMorphWithColor value: (Color yellow).
parent addMorphBack: subparent.
parent addMorphBack: (Morph new color: Color black).

parent openInWorld.
```

Wie Submorphe bei der Anzeige bezüglich ihrem Owner positioniert werden, hängt davon ab, ob die Position fest via Koordinaten festgelegt wird oder ob der Owner seine Submorphs automatisch durch ein bestimmtes Layout ausrichtet. Der parametrisierte Block `newParentMorphWithColor` erzeugt einen Morphen einer bestimmten Farbe und legt Layout-Einstellungen fest, sodass die Baumstruktur auch sinnvoll in Abbildung 2.2 nachvollzogen werden kann. `parent` ist der Owner von dem roten (`subparent`) und dem schwarzen Morph. Der rote Morph ist selbst wiederum der Owner von dem blauen und grünen Morph. Diese Baumstruktur lässt sich beliebig fortsetzen.

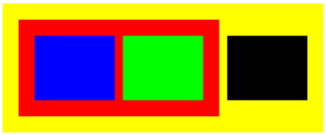


Abbildung 2.2: *Owner* und *Submorphe* in Morphic. Die gelben und roten Morphe sind *Owner* und enthalten entsprechend *Submorphe*.

Innerhalb einer Methode kann der *Owner* durch `self owner` und die *Submorphe* durch `self submorphs` referenziert werden.

2.4 FORMS UND BITBLTS

Wie in Goldberg u. Robson (1989) dokumentiert, ist der Grafik-Kernel von Smalltalk mit Hilfe von sogenannten *Forms* und *BitBlts* implementiert. Da SAM ebenfalls von *Forms* und *BitBlts* Gebrauch macht, werden diese im Nachfolgenden kurz vorgestellt.

Es sei angemerkt, dass das Morphic-Framework (siehe Abschnitt 2.3 weiter oben) ebenfalls mit *Forms* und *BitBlts* implementiert ist.

2.4.1 *Forms*

In Smalltalk wird ein Bild durch eine Instanz der Klasse `Form` repräsentiert. Mit der Klassenmethode `fromFileNamed:` kann ein Form-Objekt erstellt werden, welches die Bild-Datei repräsentiert, dessen Dateipfad übergeben wurde:

Listing 2.2: Ein Bild Anzeigen via Morphic-Framework

```
f := Form fromFileNamed: 'resource.images\Countdown0.gif'.
f asMorph openInWorld.
```

Mit der zweiten Zeile in dem Listing wird das Form-Objekt in einen Morph (siehe Abschnitt 2.3 weiter oben) umgewandelt, um das Bild anzuzeigen. Wenn die Anforderung gestellt wird, ein Bild möglichst schnell auf dem Display zu zeichnen, dann ist der „Umweg“ über das Morphic-Frameworks relativ teuer. Insbesondere in SAM, in dem ein Versuchsschritt durch eine Folge von Simulatonsschritten der Dauer 39ms realisiert wird (siehe Abschnitt 2.5 weiter unten), ist die Benutzung des Morphic-Frameworks problematisch aufgrund der Langsamkeit.

2.4.2 *BitBlts*

Alle grafischen Objekte in Smalltalk werden verändert und erstellt durch BitBlt-Objekte (auch Morphs). BitBlt-Objekte kapseln die auszuführenden Manipulationen auf Form-Objekten. Die „Ausführung“ dieser Objekte wird durch die Hardware beschleunigt durchgeführt. Die BitBlt-Klasse ist für sehr allgemeine Operationen ausgelegt, sodass relativ viele Parameter beim Erstellen eines BitBlt-Objekts angegeben werden müssen:

Listing 2.3: Ein Bild Anzeigen (via BitBlt)

```
f := Form fromFileNamed: 'resource.images\Countdown0.gif'.
bitBlt := BitBlt destForm: Display
  sourceForm: f
  halftoneForm: nil
  combinationRule: Form blend
  destOrigin: 0@0
  sourceOrigin: 0@0
  extent: f extent
  clipRect: (Rectangle origin: 0@0 extent: f extent).
bitBlt copyBits.
```

Dieses Listing ist zu dem aus Abschnitt 2.4.1 äquivalent. Auch hier wird das gewünschte Bild angezeigt, allerdings bedeutend schneller. Im Wesentlichen wird ein Bild, repräsentiert durch eine Form (Parameter sourceForm im Listing), auf ein anderes Bild (auch eine Form, Parameter destForm im Listing) kopiert. Bei Display handelt es sich ein globales Form-Objekt, welches den angezeigten Bildschirminhalt repräsentiert. Das Erzeugen des BitBlt-Objekts kapselt nur die auszuführende Operation. Erst durch Aufruf von copyBits erfolgt die Ausführung.

2.5 SAM

Wie in Abschnitt 1.1 weiter oben angemerkt, ist SAM in Bothe u. a. (2009) dokumentiert. An dieser Stelle sollen die für diese Arbeit relevanten Aspekte in den folgenden Abschnitten hervorgehoben werden.

2.5.1 *Versuch, Versuchsschritt und Simulationsschritt*

SAM ermöglicht es „rechnergestützte Versuche durchzuführen, um die Funktionsteilung zwischen Mensch und Maschine untersuchen zu können“[8].

Ein „Versuch“ besteht aus mehreren „Versuchsschritten“. Ein Versuchsschritt legt die Strecke, die Hindernisse für die Strecke und die Art der Verarbeitung der Joystick-Eingaben der beiden MWB fest.

Ein Versuchsschritt wird durch die Ausführung von „Simulationsschritten“ realisiert. Ein Simulationsschritt kann als eine Verarbeitungseinheit des Versuchsschritts betrachtet werden. Diese Einheit ist idealerweise 39ms lang. Folgende Operationen werden in einem Simulationsschritt durchgeführt (grob):

1. Einlesen der Joystick-Eingaben der MWB.
2. Verarbeitungseinheit des AAF-Frameworks ausführen.
3. Verarbeiten der Joystick-Eingaben der MWB (Aktualisierung der Position des Objekts auf der Strecke).
4. Zeichnen des aktuellen Streckenabschnitts.
5. Zeichnen der Hindernisse, falls welche vorhanden.
6. Zeichnen des Objekts.
7. Log-Eintrag schreiben.

Technisch betrachtet entspricht die Verarbeitungseinheit einer Iteration der Hauptschleife in `SAMControllerStep processStep`.

2.5.2 *Gabelungen*

Auf den Strecken können unterschiedliche Gabelungen positioniert sein. Insgesamt sind vier verschiedene Gabelungstypen vorhanden, wovon nur die beiden in Abbildung 2.3 gezeigten benutzt werden. „RLl.bmp“ und „ELr.bmp“ sind die zugehörigen Bild-Dateien für die Streckenkacheln der dargestellten Gabelungen. In dieser Arbeit seien die beiden Gablungen als „runde Gabelung“ und „eckige Gablung“ bezeichnet.

8 vgl. Bothe u. a. (2009), S. 5

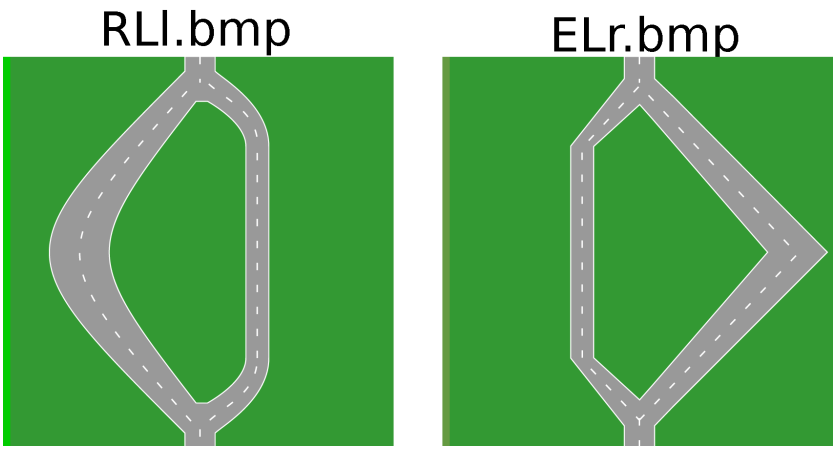


Abbildung 2.3: Links die „runde“ und rechts die „eckige“ Gabelung.

2.5.3 *Hindernisse*

In SAM wird zwischen dynamischen und statischen Hindernissen unterschieden. Bei den statischen Hindernissen kann weiter nach Abdeckung der Fahrbahn (25%, 50%) differenziert werden. Die statischen Hindernisse treten stets als Slalom auf. Das erste statische Hindernis des Slalom belegt stets die linke Fahrspur, das zweite die rechte Fahrspur. Das dynamische Hindernis bewegt sich von links nach rechts auf die Fahrbahn zu, „wobei die Geschwindigkeit des Hindernisses auf das Fahrverhalten der MWBs so angepasst ist, dass ohne Änderung des Fahrverhaltens, sprich Beschleunigen oder Bremsen, eine Kollision unausweichlich ist.“[9].

Abbildung 2.4 zeigt zwei statische sowie ein dynamisches Hindernis.

Insgesamt gibt es demnach den „statischen Hindernisslalom mit 25%-Abdeckung der Fahrbahn“, den „statischen Hindernisslalom mit 50%-Abdeckung der Fahrbahn“ und das „dynamische Hindernis“[10].

2.6 IMPLEMENTIERUNG VON FUNKTIONEN IM AAF

Mit der AAF-Komponente kann Einfluss auf SAM ausgeübt werden. Automatik-Funktionen werden im Kontext von AAF als Agenten aufgefasst. Die Implementierung neuer Agenten ist in Fuhrmann (2010), Anhang F dokumentiert. Die für diese Arbeit relevanten Informationen werden im Folgenden wiedergegeben.

9 vgl. Bothe u. a. (2009), S. 19

10 Ist das erste Hindernis auf der rechten Fahrspur positioniert und das zweite auf der linken, können zwei weitere Slaloms unterschieden werden. Das ist intern umgesetzt, wird aber praktisch nicht benutzt.

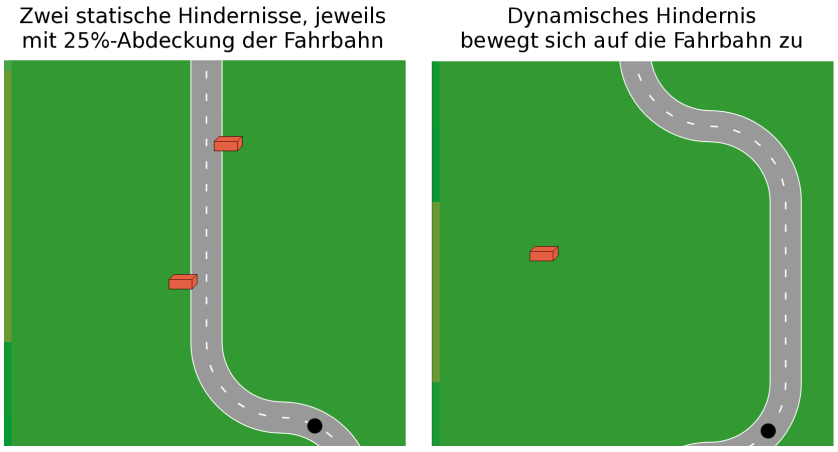


Abbildung 2.4: Statische und dynamische Hindernisse.

Jeder Agent wird durch eine Klasse implementiert, die von AAFAgent abgeleitet wird. In der Klasse muss die Methode compute : aSamState implementiert werden:

Listing 2.4: AAFAgent compute:

```
NeuerAgent compute: aSamState
  ...
  ^ aSamState
```

Bei dem aSamState handelt es sich um ein Objekt, welches den relevanten[11] Zustand von SAM für den aktuellen Simulationsschritt repräsentiert. Durch Veränderung des aSamState kann der Agent den Simulationsschritt beeinflussen. Zum Beispiel könnten die Joystick-Eingaben manipuliert werden, um Lenk-Korrekturen vorzunehmen. Der veränderte aSamState muss als Rückgabewert zurückgegeben werden.

2.7 IST-ZUSTAND

In diesem Abschnitt wird der vorgefundene IST-Zustand von SAM bezüglich der auditiven und visuellen Hinweisen beschrieben.

Wie in Abschnitt 1.1 erwähnt, kann über den Operateursarbeitsplatz auch in den Versuchsablauf eingegriffen werden. Darunter fällt auch die Anzeige von visuellen Hinweisen und das Abspielen von auditiven Hinweisen. Diese Funktionalität auf Seiten von SAM ist im Wesentlichen in den Klassen SAMControllerOperator und SAMViewOpearor implementiert. Für jeden Simulationsschritt werden entsprechende Methoden aufgerufen, die überprüfen, welche Hinweise gegeben werden sollen.

11 Aus Performance-Gründen enthält aSamState nicht alle Variablen, die den Zustand von SAM für einen Simulationsschritt beschreiben würden, sondern nur die, die bisher für die Agenten benötigt wurden.

- *Visuelle Hinweise*
  Für die visuellen Hinweise werden die in Abschnitt 2.4.2 vorgestellten BitBlt-Objekte zur Anzeige von Bildern benutzt.

- *Auditive Hinweise*
  Für die auditiven Hinweise werden StreamingMP3Sound-Objekte benutzt. Mit der Klassenmethode onFileNamed: aPath wird ein Objekt erzeugt, welches die MP3-Datei mit dem Pfad aPath abspielen kann. Schließlich kann mit der Methode play der Sound asynchron abgespielt werden.

3 THEORETISCHE AUSEINANDERSETZUNG

In diesem Kapitel werden einerseits Prozess-Modelle der Software-Entwicklung betrachtet (Abschnitt 3.1) und andererseits wird auf die Gestaltung von und Interaktion mit Benutzungsschnittstellen (Abschnitt 3.2) eingegangen.

3.1 PROZESS-MODELLE DER SOFTWARE-ENTWICKLUNG

Der Software-Entwicklung wird üblicherweise ein Prozess-Modell zu Grunde gelegt, welches den organisatorischen Rahmen für die Entwicklung festlegt. Nach Balzert (2000) legt ein Prozess-Modell fest, „welche Aktivitäten in welcher Reihenfolge von welchen Personen erledigt werden und welche Ergebnisse dabei entstehen und wie diese in der Qualitätssicherung überprüft werden". Das Endergebnis ist ein Software-Produkt, welches an den Kunden ausgeliefert werden kann.[1]

Die unterschiedlichen Prozess-Modelle haben nach Sommerville (2011) folgende fundamentalen Aktivitäten gemeinsam:

1. *Spezifikation*
   Die benötigte Software wird definiert: Welche Funktionalität benötigt der Kunde und welche Randbedingungen sind gegeben?

2. *Design und Implementierung (Entwicklung)*
   Die Spezifikation wird umgesetzt.

3. *Validierung*
   Es wird überprüft, ob die entwickelte Software der Spezifikation gerecht wird und ob die Erwartungen des Kunden erfüllt werden.

4. *Evolution*
   Die Software wird an neue Anforderungen des Kunden oder des Markts angepasst.

Diese Aktivitäten bestehen aus einer Menge von Unteraktivitäten. Bei der Spezifikation ist das zum Beispiel die Machbarkeitsstudie oder die Analyse zur Ermittlung der genauen Anforderungen. Neben diesen fundamentalen Aktivitäten gibt es auch noch projektbegleitende Aktivitäten wie Dokumentation und Konfigurationsmanagement. Es gibt kein ideales Prozess-Modell, sodass Organisationen teilweise ihre eigenen Prozess-Modelle entwickelt

1 vgl. Balzert (2000), S. 54

haben. Manchmal ist es möglich ein Prozess-Modell einer der beiden folgenden Kategorien zuzuordnen:

- *Planbasierte Prozess-Modelle*
  Alle Aktivitäten werden im Voraus genau geplant und der Fortschritt der Software-Entwicklung wird an der Einhaltung des Plans gemessen. Diese Prozess-Modelle werden vorrangig bei der Entwicklung von Systemen angewendet, dessen Ausfall oder Versagen z. B. zu schwerwiegenden Verletzungen oder zum Tod von Menschen führen würde.

- *Agile Prozess-Modelle*
  Die Planung erfolgt stets inkrementell, sodass es einfacher ist, mit veränderten oder neuen Anforderungen umzugehen. Daher werden diese Prozess-Modelle z. B. für die Entwicklung von E-Commerce-Anwendungen eingesetzt.

Die Prozess-Modelle dieser Kategorien schließen sich nicht aus und werden häufig gemeinsam in einem Projekt verwendet. Sind die Anforderungen für ein Teil-System schon am Anfang sehr klar, kann die Entwicklung des Teil-Systems dem planbasierten Prozess-Modell nach erfolgen. Sind die Anforderungen für ein Teil-System jedoch noch unbekannt oder sehr vage vom Kunden formuliert (z. B. für Benutzungsschnittstellen), empfiehlt sich die Anwendung eines agiles Prozess-Modell.

In den beiden folgenden Unterabschnitten werden zwei Prozess-Modelle, je eins aus einer der oben vorgestellten Kategorien, vorgestellt.[2]

3.1.1 *Wasserfall-Modell*

Das Wasserfall-Modell ist ein Beispiel für ein planbasiertes Prozess-Modell.

Beim Wasserfall-Modell werden verschiedene Phasen definiert, in denen bestimmte Artefakte (vor allem Dokumente) erzeugt werden, die der nachfolgenden Phase als Eingabe dienen. Eine Phase wird erst begonnen, nachdem die vorherige Phase vollständig abgeschlossen wurde. In der Praxis sind die Aktivitäten der einzelnen Phasen jedoch überlappend, da Fehler in einer Phase erst in der nachfolgenden Phase festgestellt werden können. Ein Fehler in einer der frühen Phase (z. B. Spezifikation) kommt entsprechend sehr teuer zur Geltung, wenn dieser erst in einer der späten Phasen entdeckt wird. Die Anpassung der einzelnen Artefakte ist sehr aufwändig, sodass in der Praxis z. B. nach ein paar Iterationen die Dokumente vernachlässigt werden. Projekte mit wechselnden Anforderungen sind daher ungeeignet für dieses Prozess-Modell.[3]

2 vgl. Sommerville (2011), S. 27ff.
3 vgl. Sommerville (2011), S. 30ff.

3.1.2 *Inkrementelle Entwicklung*

Die Inkrementelle Entwicklung ist ein Beispiel für ein agiles Prozess-Modell.

Die Spezifikation, Entwicklung und Validierung sind bei diesem Modell verzahnt. Dem Kunden wird so früh wie möglich eine erste Version des funktionierenden Systems gezeigt. Dieser kommentiert das System, sodass die veränderten oder neuen Anforderungen für die nächste Version umgesetzt werden. In jeder Version kommt demnach neue Funktionalität hinzu. Dieser Vorgang wiederholt sich, bis der Kunde mit dem gewünschten System zufrieden ist. Durch das frühe Feedback vom Kunden sind veränderte Anforderungen nicht so fatal wie beim Wasserfall-Modell. Für den Kunden ist vorteilhaft, dass er ein lauffähiges System zu Gesicht bekommt, anstatt Dokumente. Außerdem kann eine vorgestellte Version vom Kunden früher eingesetzt werden. Problematisch an der inkrementellen Entwicklung ist, dass die Systemstruktur von Version zu Version schwieriger zu durchschauen wird, wenn nicht genug Zeit für Restrukturierungen gegeben wird.[4]

3.2 GESTALTUNG VON UND INTERAKTION MIT BENUTZUNGSSCHNITTSTELLEN

Die Benutzungsschnittstelle erlaubt es dem Benutzer alle Daten und Funktionen einzusehen, um die notwendigen Geschäftsprozesse durchzuführen.[5]

Um im Rahmen dieser Arbeit eine möglichst gute Benutzungsschnittstelle zu entwickeln, wird im Folgenden auf die Software-Ergonomie (Abschnitt 3.2.1), die Gebrauchstauglichkeit (Abschnitt 3.2.2) und die Grundsätze der Dialoggestaltung (Abschnitt 3.2.3) eingegangen.

3.2.1 *Software-Ergonomie*

Nach Dahm (2006) ist *Ergonomie* die Lehre von der menschlichen Arbeit, ihrer Beschreibung, Modellierung und Verbesserung. Die Ergonomie stellt dabei nicht die Arbeit in den Vordergrund, sondern den Menschen, seine (Arbeits-)Aufgaben und seine Bedürfnisse. Dabei geht es nicht nur um den Schutz vor negativen Einwirkungen (Verletzungen, unzumutbaren Arbeitsbedingungen, psychische Einwirkungen wie z. B. Stress, Anpassung oder Überforderung), sondern auch darum, den Menschen bei seiner Arbeit zu unterstützen, zu fördern und zufrieden zu stellen.[6]

4 vgl. Sommerville (2011), S. 32ff.
5 vgl. Denert (1992), S. 125f.
6 vgl. Dahm (2006), Abschnitt 2.1, S. 28

Übertragen auf Software, bedeutet *Software-Ergonomie* nach Heinecke (2004) das Streben nach der Anpassung der Eigenschaften eines Software-Systems an die physischen und psychischen Eigenschaften der damit arbeitenden Menschen.[7]

3.2.2 *Gebrauchstauglichkeit und ihre Leitziele*

Im Rahmen der Software-Ergonomie ist der Begriff der Gebrauchstauglichkeit (engl. Usability) von zentraler Bedeutung. Die Gebrauchstauglichkeit wird in der EN ISO 9241-11 definiert:

> Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.

Die Gebrauchstauglichkeit ist also abhängig von den Benutzern, ihren Zielen und dem Nutzungskontext. Eine Software, die von anderen Benutzern als intendiert eingesetzt wird, würde von diesen höchstwahrscheinlich mit einer geringen Gebrauchstauglichkeit bewertet werden. Aus diesem Grund ist es bei der Software-Entwicklung sehr wichtig, die zukünftigen Benutzer zu kennen.

Effektivität, Effizienz und Zufriedenheit seien nochmal genauer betrachtet:

EFFEKTIVITÄT *Die Genauigkeit und Vollständigkeit, mit der Benutzer ein bestimmtes Ziel erreichen.*

EFFIZIENZ *Der im Verhältnis zur Genauigkeit und Vollständigkeit eingesetzte Aufwand, mit dem Benutzer ein bestimmtes Ziel erreichen.*

ZUFRIEDENHEIT *Freiheit von Beeinträchtigungen und positive Einstellung gegenüber der Nutzung des Produkts.*

Ist eine Software effektiv, dann bietet sie dem Benutzer alle notwendigen Funktionen, um seine Aufgabe zu erfüllen. Ist die Software zusätzlich effizient, dann kann der Benutzer sein Ziel mit relativ geringem Aufwand, z. B. in kurzer Zeit, erreichen. Damit eine Software den Benutzer zufrieden stellt, muss sie auf ihn und seine Aufgaben zugeschnitten sein.

3.2.3 *Grundsätze der Dialoggestaltung*

In der Norm „DIN EN ISO 9241: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten“ werden in Teil 110 die Grundsätze der Dialoggestaltung beschrieben. Ein Dialog wird definiert als

7 vgl. Heinecke (2004), Abschnitt 2.1.2, S. 40

> „Interaktion zwischen einem Benutzer und einem interaktiven System in Form einer Folge von Handlungen des Benutzers (Eingaben) und Antworten des interaktiven Systems (Ausgaben), um ein Ziel zu erreichen."

Dazu zählen auch navigierende Handlungen des Benutzers. Die Grundsätze sind allgemein gehalten und sind somit unabhängig von „spezieller Technologie oder Technik". Sie werden im Folgenden zitiert und die wichtigen Aspekte werden hervorgehoben.

AUFGABENANGEMESSENHEIT *Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen.*

Hierzu zählt u. a. dass nur relevante Informationen angezeigt werden, Ein- und Ausgaben beispielsweise regionalen oder nationalen Konventionen entsprechen (z. B. Währungen), sinnvolle Voreinstellungen getroffen werden und dass keine unnötigen Dialogschritte[8] vom Benutzer gemacht werden müssen. Es sind vernünftige Voreinstellungen zu wählen und wiederkehrende Aufgaben sollte das System unterstützen.

SELBSTBESCHREIBUNGSFÄHIGKEIT *Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird.*

Beispielsweise sollte der Benutzer während des Dialogs möglichst nicht eine Information erst extern nachschlagen müssen. Für numerische Eingaben sollte demnach die Einheit angezeigt werden. Dem Benutzer sollte klar sein, wie viele Schritte im Dialog noch vor ihm liegen und es sollte eine einheitliche Terminologie verwendet werden.

ERWARTUNGSKONFORMITÄT *Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z. B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung sowie den allgemein anerkannten Konventionen.*

Der Benutzer sollte möglichst nicht (negativ) überrascht werden. Hierzu gehört, dass sprachliche und kulturelle

8 In Cooper u. a. (2010) werden diese unnötigen Dialogschritte als „Rüstaufgaben" bezeichnet. Dazu zählt z. B. das Manipulieren von Fenstern - es bringt den Benutzer seinem Ziel nicht näher, sondern ist nur ein notwendiger Schritt, um eine Aktion im Programm auszuführen. Zu den Rüstaufgaben zählt auch die Navigation innerhalb des Programms.

Konventionen eingehalten werden, dass das Vokabular des Benutzers Verwendung findet und dass der Benutzer eine Rückmeldung bekommt, wenn das System länger benötigt als er erwartet. Die Art und Länge von Rückmeldungen sollte angemessen sein. Konsistenz bei ähnlichen Arbeitsschritten führt ebenfalls zu erhöhter Erwartungskonformität.

LERNFÖRDERLICHKEIT *Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen der Nutzung des Dialogsystems unterstützt und anleitet.*

Der Benutzer sollte sein Ziel mit minimalem Lernaufwand erreichen. Hier sind u. a. sinnvolle Voreinstellungen, ein UNDO[9]- und ein Hilfe-System zu empfehlen. Aus Rückmeldungen und Erläuterungen sollte der unerfahrene Benutzer ebenfalls lernen können[10].

STEUERBARKEIT *Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.*

Der Benutzer sollte die Möglichkeit haben, seine Aufgabe zu unterbrechen (z. B. durch Abspeichern einer „Sitzung“) und zu einem späteren Zeitpunkt wiederaufzunehmen. Sinnvolle Voreinstellungen sollten durch den Benutzer ebenfalls veränderbar sein und bei vielen anzuzeigenden Daten sollte die angezeigte Menge regulierbar sein.

FEHLERTOLERANZ *Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.*

Als Mittel für Fehlertoleranz zählt Fehlererkennung und –vermeidung. Wenn es zu einem Fehler kommt (Fehlererkennung), sollte der Benutzer ausreichend informiert werden, um den Fehler zu beseitigen. Dazu gehört, dass der Benutzer an die fehlerhafte Stelle verwiesen wird. Fehler im Zusammenhang mit dem Wertebereich einer Eingabe können bei numerischen Eingaben beispielsweise über ein Spinbox-Widget verhindert werden. Durch die vorgegebenen Werte hat der Benutzer keine Möglichkeit eine fehlerhafte Eingabe

9 Mit einem UNDO-System kann der Benutzer durchgeführte Aktionen rückgängig machen.

10 vgl. Johnson (2008), Kapitel 1, S. 19ff: Hier wird von einem „Conceptual Model“ gesprochen. Es ist das Modell der Anwendung, die der Benutzer verstehen soll. Rückmeldungen und Erläuterungen helfen dem Benutzer dieses Modell zu erfassen.

zu tätigen. Eine automatisch durchgeführte Fehlerkorrektur sollte dem Benutzer kenntlich gemacht werden und er sollte die Möglichkeit haben, diese rückgängig zu machen.

INDIVIDUALISIERBARKEIT *Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe sowie an die individuellen Fähigkeiten und Vorlieben des Benutzers zulässt.*

Sprache sowie kulturelle Eigenheiten verlangen nach Einstellungsmöglichkeiten. Farben und Schriftgrößen sollten ebenfalls anpassbar sein. Wo möglich, sollten mehrere Wege existieren um eine Aktion auszuführen (für neue Benutzer z. B. ein Menüeintrag, für erfahrene Benutzer eine Tastenkombination). Format und Typen von Eingabe- und Ausgabeoperationen sollten ebenfalls einstellbar sein. Schließlich sollten alle Einstellungen (auf Voreinstellungen) zurückgesetzt werden können.

Diese Grundsätze überlappen sich. In einigen Fällen schließen sie sich sogar teilweise aus. Es sollte immer von der Benutzergruppe, dem Kontext und der eingesetzten Dialogtechnik abhängig gemacht werden, welcher Grundsatz vorzuziehen ist.

3.2.4 *Richtlinien*

Von den ersten Tagen an haben Designer von Benutzungsschnittstellen Richtlinien aufgeschrieben, um ihre Einsichten festzuhalten. Die frühen Richtlinien von Apple und Microsoft haben viele Designer beeinflusst und finden teilweise heute im Web und auf mobilen Geräten Anwendung.

Inzwischen gibt es eine beachtliche Anzahl von Richtlinien, Regeln, Heuristiken und Prinzipien. Obwohl sich vieles überschneidet, auch mit den Grundsätzen der Dialoggestaltung (vgl. 3.2.3), haben alle ihren Wert, da sie unterschiedliche Aspekte betonen. Die für diese Arbeit durchgesehenen Richtlinien seien im Folgenden aufgelistet.

1. „Eight golden rules of interface design“ von Shneiderman u. Plaisant (2005), Abschnitt 2.3.4
2. „Ten Usability Heuristics“ von Nielsen (2005)
3. Ergänzungen zu Nielens Heuristiken und Shneidermans acht goldenen Regeln von Dahm (2006), Abschnitt 8.4, S. 157
4. Designprinzipien von Cooper u. a. (2010), Anhang A
5. „Basic Principles“ von Johnson (2008), Kapitel 1

4 ANALYSE DER KONZEPTBÖGEN

In diesem Kapitel werden die Konzepte analysiert. Die Konzepte wurden von den Entwicklern erarbeitet und in Konzeptbögen auf Papier beschrieben und teilweise mit Skizzen ergänzt. Kain (2012) beschreibt den Entwicklungsprozess im Detail. An dieser Stelle sind folgende Informationen zum Verständnis der Analyse ausreichend:

- Insgesamt haben 30 Teams, bestehend aus je drei Mitgliedern, an der Entwicklung der Konzepte mitgewirkt.
- Alle Teams haben Erfahrung in der Entwicklung von Automatiken, Assistenzsystemen oder anderen technischen, komplexen und dynamischen Systemen. Zu den Unternehmen, aus denen die Entwickler kommen, zählen u. a. BMW, Honda, Airbus, EADS, Diehl, Nec und DLR Braunschweig.
- Den Teams wurde SAM und ihre Aufgabe vorgestellt. Danach hatten die Entwickler Zeit, um die Konzepte zu erarbeiten, diese in den Konzeptbögen zu dokumentieren und gegebenenfalls Skizzen zu erstellen. Insgesamt hatten die Teams dafür nur wenige Stunden Zeit.
- Die Vorstellung von SAM, der Aufgabe sowie der ganze Entwicklungsprozess wurden per Videoaufzeichnung festgehalten.
- Während die entstandenen Videos von Kain, angehenden Informatikern und einer studentischen Hilfskraft transkribiert wurden bzw. noch werden, hat Kain auch die Konzeptbögen und Skizzen digitalisiert.

  Die Konzeptbögen sind wie folgt strukturiert: Jedes Team entwickelt genau ein Konzept zur Unterstützung der MWB. Ein Konzept kann aus einer oder mehreren Automatik(en) bestehen. Eine Automatik wird durch eine oder mehrere Funktion(en) realisiert.

In den nachfolgenden Abschnitten wird dem Leser ein Überblick über die auditiven und visuellen Hinweise gegeben. Dabei wird die Vielfalt der Hinweise sowie die ungenaue und unvollständige Spezifikation der Konzepte deutlich gemacht. Zum Ende dieses Kapitels soll der Leser mit den hier vorgestellten Kategorisierungen der Hinweise vertraut sein.

4.1 ABGRENZUNG UND ÜBERBLICK

Für die Analyse wurden alle Konzepte durchgesehen und auf auditive und visuelle Hinweise untersucht. Es liegen insgesamt 30 Konzepte, durchnummeriert beginnend mit 20[1], vor. Insgesamt wurden 114 Automatiken und 279 Funktionen spezifiziert. 110 Funktionen realisieren auditive oder visuelle Hinweise. Von den Funktionen, die Hinweise realisieren, sollen 65 mit dieser Arbeit umgesetzt werden[2]. Nicht bearbeitet werden die Hinweise, „die das Anzeigen der aktuellen und optimalen Objektposition, -richtung und -geschwindigkeit und das Anzeigen der aktuellen und optimalen Joystickposition der MWB beinhalten“[3]. Diese speziellen visuellen Hinweise werden aktuell von Aydan Seid implementiert. Zur Abgrenzung werden im Anhang A auf Seite 117 alle Funktionen aus den Konzeptbögen aufgelistet, die durch diese Arbeit umgesetzt werden sollen.

Tabelle 4.1 dient einem Gesamtüberblick und dokumentiert, welche Konzepte auditive und visuelle Hinweise enthalten. In der Tabelle ist ebenfalls vermerkt, ob die Hinweise in den Konzepten vollständig, teilweise oder gar nicht durch diese Arbeit implementiert werden:

- Keine Markierung bedeutet, dass die Hinweise in dem Konzept vollständig durch diese Arbeit umgesetzt werden.
- * markiert ein Konzept, in dem mind. ein Hinweis vorkommt, der durch diese Arbeit umgesetzt werden soll, und mind. einen Hinweis, der von Seid bearbeitet wird.
- ** markiert ein Konzept, bei dem die Hinweise vollständig von Seid implementiert werden.

Alle 30 Teams haben sich für die Verwendung von visuellen Hinweisen ausgesprochen und haben diese in ihren Konzepten berücksichtigt. 19 von 30 Teams verwenden auch auditive Hinweise. Mit dieser Arbeit soll zu der Realisierung von 28 Konzepten beigetragen werden.

4.2 GRÜNDE FÜR HINWEISE

Die Gründe für die Verwendung von Hinweisen sind vielfältig. Allgemein dienen die Hinweise der Unterstützung der MWB. Die genauere Betrachtung hat die in Tabelle 4.2 aufgezeigten Gründe hervorgebracht. Die Gründe sind danach gruppiert, ob

1 Das erste Team sollte nicht den Eindruck haben das erste Team zu sein.
2 Siehe auch Anhang A auf Seite 117. Fünf weitere Funktionen konnten aus der allgemeinen Automatikbeschreibung abgeleitet werden. Diese gingen jedoch nicht explizit als Funktion aus den Konzeptbögen hervor.
3 vgl. Seid (2012)

K	AUD.	VIS.	K	AUD.	VIS.
20	Ja	Ja	35	Nein	Ja*
21	Nein	Ja**	36	Ja	Ja
22	Ja	Ja	37	Ja	Ja
23	Ja	Ja*	38	Ja	Ja*
24	Ja	Ja*	39	Ja	Ja*
25	Nein	Ja	40	Nein	Ja
26	Ja	Ja*	41	Nein	Ja
27	Nein	Ja	42	Ja	Ja
28	Nein	Ja*	43	Ja	Ja
29	Ja	Ja*	44	Ja	Ja
30	Ja	Ja*	45	Nein	Ja
31	Ja	Ja	46	Ja	Ja
32	Ja	Ja**	47	Ja	Ja
33	Nein	Ja	48	Nein	Ja*
34	Ja	Ja*	49	Nein	Ja
Summe:	10/15	15/15	Summe:	9/15	15/15

Tabelle 4.1: Analyse - Konzepte (K), die auditive (AUD.) und visuelle (VIS.) Hinweise vorsehen (** - ausschließlich von Seid zu implementieren, * - von Seid und dem Verfasser zu implementieren)

ihre Hinweise als präventive Hinweise verstanden werden können oder nicht (erste Spalte). Unter präventiven Hinweisen seien Warnungen und Vorschläge gemeint, die MWB auf eine zeitnah eintretende Situation vorbereiten oder für diese instruieren. Innerhalb der Gruppen sind die Verwendungsmöglichkeiten (zweite Spalte) nach Anzahl der Konzepte, die davon Gebrauch machen, sortiert (dritte Spalte).

Die überwiegende Anzahl an Hinweisen wird dafür eingesetzt, bei einer Gabelung einen Richtungsvorschlag anzuzeigen. Danach folgen die Hinweise, die die MWB auffordern, ihre Geschwindigkeit entsprechend der Strecke anzupassen. Die Hinweise für den Richtungsvorschlag unterstützen primär den MWB, der das genaue Fahren als Priorität verfolgt. Dagegen unterstützen die Geschwindigkeitshinweise primär den MWB, der die Priorität verfolgt, so schnell wie möglich zu fahren.

4.3 EIGENSCHAFTEN DER HINWEISE

In den folgenden Unterabschnitten werden die Eigenschaften der auditiven und visuellen Hinweise aus den Konzeptbögen

PRÄV.	VERWENDUNGSMÖGLICHKEIT	KONZEPTE
Ja	Richtungsvorschlag an Gabelungen	Alle bis auf 21, 32, 33, 39, 44
Ja	Warnung vor Hindernissen	25, 27, 30, 34, 37, 43, 44
Ja	Kollisionsvorhersage für Hindernisse	24, 29, 39, 42
Ja	Route zum Umfahren eines Hindernisses	20, 25, 37, 48
Ja	Wahrung des Abstands zum Fahrbahnrand	22, 23, 28, 43
Ja	Warnung vor Kurven	37, 43, 46
Ja	Warnung vor Gabelungen	43, 44
Ja	Streckenvorschau	37, 44
Nein	Anpassung der Geschwindigkeit	20, 22, 23, 31, 32, 37, 38, 40, 44, 45, 46, 47, 49
Nein	Anzeige der Aktivität einer Automatik	23, 25, 40, 41
Nein	Hervorhebung von Hindernissen	22, 23
Nein	Anzeige von Personenausfall oder Fehlverhalten	40

Tabelle 4.2: Analyse - Übersicht der Verwendungsmöglichkeiten von Hinweisen

vorgestellt. Es wird auch betrachtet, zu welchen Zeitpunkten die Hinweise angezeigt bzw. abgespielt werden müssen.

4.3.1 *Eigenschaften der auditiven Hinweise*

Auditive Hinweise lassen sich abstrakten Tönen oder gesprochenem Text zuordnen.

Es folgt eine Übersicht der gewünschten auditiven Hinweise. Es werden die Automatiken und Funktionen in den Konzeptbögen referenziert, die von den Hinweisen Gebrauch machen. Bei einigen Automatiken wird ein Hinweis nur in der allgemeinen Automatikbeschreibung erwähnt, ohne diesen als eigene Funktion wieder aufzugreifen. In solchen Fällen fehlt der Funktionsname in der Referenzierung.

Abstrakte Töne

- „Warnton“
  Konzept 22, Automatik „Spurhalteassistent“;
  Konzept 23, Automatik „Gefahrenwarnung & -vermeidung“
- „Piepton“, der seine Frequenz abhängig von einer Distanz zu einem Streckenelement ändert
  Konzept 24, Automatik/Funktion „Hinderniserkennung“
- „dauerhafte[r] ’unangenehme[r]’ Signalton (1k Hz) [sic]“
  Konzept 24, Automatik „Farbänderung Kreisfläche“, *namenlose Funktion*
- „Ton[,] welcher mit Ufo in Verbindung gebracht wird“
  Konzept 30, Automatik „Navigation“, Funktion „Warner“
- „einfach ein Ton“
  Konzept 34, Automatik „Ausgabe der Führungsgrößen“, Funktion „Warnsignal“
- „Bing“
  Konzept 38, Automatik „Steering Assistanz - Command Generator“, Funktion „Speed Advisor fixed obstacle“
- „Hupen“
  Konzept 39, Automatik „Notfall“
- „3 maliges [sic] Tuten“
  Konzept 42, Automatik „Warnung bei potentieller Kollision (5)“, Funktion „Kollision Warnung aural“

Gesprochener Text

- „Warnung ’BREMSEN’ + auditiv“
  Konzept 20, Automatik „Vermeiden von Kollision mit dynamischem Hindernis“, Funktion „Warnung "BREMSEN" + auditiv“
- „zu langsam“, „zu schnell“, „nach links“, „nach rechts“
  Konzept 32, Automatik „Mediator“, Funktion „Audioanweisung“
- „schneller“, „langsamer“
  Konzept 26, Automatik „dynamisches Hindernis“;
  Konzept 37, Automatik „Steuerung vorschlagen“, Funktion „’schneller’ durchsagen“ und „’langsamer’ durchsagen“ und „eines von beidem durchsagen“
- „links“, „rechts“
  Konzept 26, Automatik „Gabelung“;

Konzept 31, Automatik „Gabelungsassistent“, Funktion „Alarm“;
Konzept 37, Automatik „Gabelungsentscheidung“, Funktion „Signale an den Fahrer“;
Konzept 42, Automatik „Hinweis bei Gabelung“, Automatik „Wegweiser bei Gabelung“;
Konzept 46, Automatik „XY-Longi-Lati-Control“, Funktion „Richtungsvorgabe“;
Konzept 36, Automatik „Gabelungsentscheidung“, Funktion „Signale an den Fahrer“

- „Folgen Sie dem Pfeil“
  Konzept 43, Automatik „Gabelungsautomatik“, Funktion „Links-Rechts-Hinweis-Funktion“

- „Achtung Hindernis“, „Achtung Gabelung“, „Achtung Kurve“
  Konzept 43, Automatik „Warnanzeige für scharfe Kurven / Gabelungen und Hinderniss“, Funktion „Warnhinweis Kurve, Hindernis, Gabelung“

- „Vorsicht Gabelung“, „Vorsicht Hindernis“
  Konzept 44, Automatik „Akkustik-Assistent [sic]“, Funktion „Akkustik-Assistent [sic]“

- "Bitte denken Sie daran, den Kurs so schnell wie möglich zu absolvieren.", "Bitte denken Sie daran, den Kurs so genau wie möglich zu absolvieren."
  Konzept 44, Automatik „Moderationsassistent“, Funktion „Moderationsassistent“

4.3.2 *Eigenschaften der visuellen Hinweise*

Nach der Durchsicht der gewünschten visuellen Hinweise wurde folgende Einteilung dieser vorgenommen:

1. *Statische Hinweise* (Anzeige ändert sich nicht)
   Diese visuellen Hinweise beschränken sich auf das Einblenden von Symbolen oder statischem Text zu bestimmten Zeitpunkten. Die eingeblendete Information verändert sich von einem Simulationsschritt zum nächsten nicht.

2. *Dynamische Hinweise* (Anzeige ändert sich)
   Diese visuellen Hinweise gehen über das einfache Einblenden von Symbolen oder statischem Text hinaus. Sie sind dynamisch in dem Sinne, dass sich die Anzeige mit fortschreitender Zeit, auch über mehrere Simulationsschritte, anpasst bzw. verändert.

In einigen Konzepten wurde auch beschrieben, „wie“ die Anzeige erfolgen soll (z. B. blinkend oder halbtransparent). Diese besonderen Eigenschaften der Anzeige werden hier ebenfalls betrachtet.

Statische visuelle Hinweise

Die meisten visuellen Hinweise können als „statische Hinweise“ eingestuft werden. Es handelt sich dabei um 44 Funktionen. Die 17 Funktionen, bei denen nicht beschrieben ist, wie die Anzeige erfolgen soll, lassen sich ebenfalls den statischen Hinweisen zuordnen. Demnach können insgesamt 61 Funktionen als statische Hinweise realisiert werden.

Zu den statischen Hinweisen gehören die Warnungen vor Hindernissen und Gabelungen und der Richtungsvorschlag bei Gabelungen selbst, ebenso die Kollisionsvorhersage, das Anzeigen einer Route zum Umfahren von statischen Hindernissen sowie die Hervorhebung der Hindernisse.

Dynamische visuelle Hinweise

Hier folgt ein Überblick der dynamischen Hinweise. Insgesamt sind 14 Funktionen als dynamische visuelle Hinweise eingestuft.

- Richtungsvorschlag bei Gabelung
  Wie in Abschnitt 4.2 festgestellt wurde, soll meistens ein Richtungsvorschlag bei einer Gabelung gemacht werden. Da ein Team den Richtungsvorschlag vom Zufall abhängig macht, handelt es sich hierbei auch um einen dynamischen Hinweis (bei der ersten Gabelung wird vielleicht der linke Zweig empfohlen, bei der zweiten Gabelung der rechte).

- Geschwindigkeitshinweis
  Bei den meisten Geschwindigkeitshinweisen, insbesondere im Zusammenhang mit dynamischen Hindernissen, müssen zwei Grafiken angezeigt werden: Eine für das Erhöhen der Geschwindigkeit, eine für das Verringern dieser.

- Genauigkeitshinweis
  Bei den Genauigkeitshinweisen ändert sich auch die Anzeige über mehrere Simulationsschritte, abhängig vom Fahrverhalten der MWB: Team 22 möchte den Fahrbahnrand farbig hervorheben, je nachdem ob nur Gefahr festgestellt wird (gelb) oder die Automatik in die Steuerung eingreift (rot). Team 23 möchte umso intensiver warnen, je näher man dem Rand kommt. Team 43 möchte abhängig vom Abstand zum Fahrbahnrand drei Schilder einblenden.

- Route zum Umfahren des dynamischen Hindernisses
  Bei Konzept 48 ist eine Route zum sicheren Umfahren eines

dynamischen Hindernisses gewünscht. Da sich das dynamische Hindernis bewegt, ebenso wie das von den MWBn gesteuerte Objekt, muss diese Route ständig neu berechnet und angezeigt werden.

- Kartenvorschau
  Die Kartenvorschau, von Team 37 und 44 gewünscht, muss ständig neu aktualisiert werden und ist damit als ein dynamischer Hinweis einzustufen.

Besondere Eigenschaften der Anzeige

Einige wenige Teams wollen die Hinweise auf der Strecken verankert, blinkend oder halbtransparent angezeigt haben. Der Tabelle 4.3 kann entnommen werden, welche besonderen Eigenschaften von welchen Teams gestellt werden.

EIGENS.	KONZEPT	ZWECK
Verankerung	z. B. 20, 22, 24 (und 13 andere)	z. B. Pfeil bei Richtungsvorschlag an einer Gabelung; Fixe Route zum Umfahren von statischen Hindernissen, Hervorhebung von statischen Hindernissen
Blinken	26	Durch Blinken Hervorhebung von z. B. Geschwindigkeitsbalken
	36	Einblendung eines blinkenden Pfeils
	39	Blinken bei Kollisionsgefahr
	46	Anzeige für Geschwindigkeitsempfehlung soll blinken
	49	Blinkender Pfeil für Richtungsvorschlag an einer Gabelung
Transparenz	46	Einblendung eines halbtransparenten Rechteckes
	37, 44	Einblenden einer halbtransparenten Streckenvorschau

Tabelle 4.3: Besondere Eigenschaften visueller Hinweise

Viele Teams haben auch konkrete Vorstellungen, an welcher Position ihr Hinweis auf dem Bildschirm erscheinen soll. Bei

der Platzierung der Hinweise können zwei Arten unterschieden werden:

- *Einblenden ohne Verankerung*
  Hierbei sind die Koordinaten des Hinweises relativ zum Koordinatenursprung des Displays festgelegt. Der Hinweis wird also an einer bestimmten Position des Bildschirms eingeblendet und verändert seine Position bis zum Ausblenden nicht. Die Strecke wird in diesem Fall unter dem Hinweis „weiterlaufen". Diese Einblendung entspricht einem Head-Up-Display, wie sie z. B. bei Kampfflugzeugen üblich ist.

- *Einblenden mit Verankerung*
  Hierbei sind die Koordinaten des Hinweises relativ zur Position eines Streckenelements festgelegt. Der Hinweis wird also an einer bestimmten Position der Strecke eingeblendet und gleitet mit fortschreitender Strecke mit, bis er aus dem sichtbaren Bereich verschwindet.

Ein Beispiel für das Einblenden ohne Verankerung ist die Geschwindigkeitsempfehlung (z. B. ein Pfeil nach oben, der empfiehlt, schneller Fahren). Fahren die MWB zu langsam, wird der Hinweis solange eingeblendet, bis die Geschwindigkeit angepasst wurde, unabhängig vom Streckenabschnitt. Die Einblendung ist nur abhängig vom Fahrverhalten der MWB.

Ein Beispiel für die Einblendung mit Verankerung ist ein Richtungsvorschlag an einer Gabelung. Das entspricht einem Schild im Straßenverkehr.

4.3.3 *Aktivierung der Hinweise*

Bisher wurden die Eigenschaften von auditiven und visuellen Hinweisen betrachtet. Ebensoviel Aufmerksamkeit muss der Frage beigemessen werden, zu welchem Zeitpunkt die Hinweise abgespielt bzw. angezeigt werden sollen und wie lange.

Es lässt sich zwischen drei Kategorien von Situationen unterscheiden:

1. Situationen, die auf Streckenelementen basieren
2. Situationen, die auf dem Fahrverhalten der MWB basieren
3. Situationen, die auf Streckenelementen und dem Fahrverhalten der MWB basieren

Es wird betrachtet, welche Situationen aus den Konzeptbögen hervorgehen.

Situationen basierend auf Streckenelementen

- *Gabelungen* (27 Konzepte)
  ...für Hinweise mit Warnfunktion (z. B. Konzept 43) und Hinweise, die einen Richtungsvorschlag an Gabelungen machen (z. B. Konzept 22)

- *Dynamische Hindernisse* (19 Konzepte)
  ...für Hinweise mit Warnfunktion (z. B. Konzept 34), aber auch welche, die dann eine Geschwindigkeitsempfehlung machen (z. B. Konzept 37)

- *Statische Hindernisse* (11 Konzepte)
  ...für Hinweise, die statische Hindernisse hervorheben möchten (z. B. Konzept 22) oder eine Route zum Umfahren dieser Hindernisse einblenden (z. B. Konzept 48)

- *Kurven, abgestuft nach verschiedenen Krümmungen* (3 Konzepte)
  ...für Hinweise mit Warnfunktion (z. B. Konzept 37)

Situationen basierend auf dem Fahrverhalten

- MWB *haben durch Knopfdruck eine bestimmte Automatik aktiviert* (1 Konzept)
  ...für einen bestimmten Hinweis, der die Aktivität einer Automatikfunktion anzeigt (Konzept 40)

- *Personenausfall* (1 Konzept)
  ...für einen bestimmten Hinweis, der den detektierten Personenausfall anzeigt (Konzept 40)

- *Zu langsames, zu schnelles Fahren* (9 Konzepte)
  ...für Hinweise, die die MWB bei möglichst schnellem (und genauem) Fahren unterstützen (z. B. Konzept 44);

- *Zu ungenaues Fahren* (1 Konzept)
  ...ebenfalls für Hinweise, die die MWB bei möglichst genauem Fahren unterstützen (Konzept 44)

- *Abstand zum Fahrbahnrand (in verschiedenen Abstufungen) und Abstandsänderungsgeschwindigkeit* (4 Konzepte)
  ...für Hinweise, die die MWB bei möglichst genauem Fahren unterstützen (z. B. Konzept 43)

4.3.3.1 *Situationen basierend auf Streckenelementen und dem Fahrverhalten*

Neben den oben aufgelisteten Situationen gibt es eine Menge Situationen, die sowohl auf Streckenelementen beruhen, aber auch das Fahrverhalten der MWB miteinbeziehen.

- *Kollisionskurs mit Hindernissen* (8 Konzepte)
  ...ist nur sinnvoll in Kombination mit den Situationen „dynamisches Hindernis“ und „statisches Hindernis“ (z. B. Konzept 22)

- MWB *tendieren vor Gabelungen zu linkem oder rechtem Zweig* (1 Konzept)
  ...für den Richtungsvorschlag an Gabelungen von Konzept 43. Dieser kommt der Lenktendez der beiden MWB entgegen und schlägt den entsprechenden Gabelungszweig vor

- *Differenz der Joystick-Eingaben der beiden* MWB *ist zu klein/groß* (3 Konzepte)
  ...für Hinweise, die Richtungsvorschläge an Gabelungen machen oder einen Konflikt bzw. ein Fehlverhalten feststellen möchten (z. B. Konzept 40)

4.4 EMPFEHLUNG EINES GABELUNGSZWEIGS

Besonders variationsreich sind die Angaben der Entwickler bei der Empfehlung eines Gabelungszweigs. Aus diesem Grund werden die Möglichkeiten hier genauer betrachtet.

25 Konzepte möchten bei Gabelungen einen Zweig empfehlen. 9 Konzepte lassen offen, welcher Zweig gewählt werden soll. Die übrigen 16 Konzepte haben dazu eine Angabe gemacht, die Tabelle 4.4 auf der nächsten Seite entnommen werden kann. Die Zeilen sind nach Ähnlichkeit der Empfehlung gruppiert.

4.5 UNGENAUIGKEITEN UND FEHLENDE INFORMATIONEN IN DEN KONZEPTEN

Viele Konzepte beinhalten ungenaue oder unvollständige Beschreibungen. Das ist auf die kurze Entwicklungszeit von nur zwei Stunden zurückzuführen.

Da die meisten Konzepte Ungenauigkeiten enthalten, wäre es nicht sinnvoll, diese aufzulisten. Stattdessen werden einige Ungenauigkeiten hervorgehoben, um dem Leser ein Gefühl des Grads an Ungenauigkeit zu vermitteln. Der allgemeine Umgang mit fehlenden Informationen wird in Abschnitt 5.1 geschildert. Die Behandlung spezieller Ungenauigkeiten wird in Kapitel 6 beschrieben.

Die Konzeptbögen von Kain (2012) enthalten eine Spalte „Beschreibung“, in welcher die auditiven und visuellen Hinweise noch am genauesten beschrieben werden. Folgende Ungenauigkeiten können u. a. in dieser Spalte festgestellt werden:

- Es gibt Teams (z. B. 23, 39 und 45), die visuelle Hinweise verwenden möchten, aber keine weiterführenden Angaben

KONZEPT	ZWEIGEMPFEHLUNG
23, 42	„rechts“, „immer rechten Zweig“
35	„dünnere[r] Ast“
37	„breiter Zweig“
36, 45	„kürzere Gabelung“, „Kürzerer Zweig“
49	„leichteren Zweig wählen, wenn beide gleich, dann zufällig!“
20, 26, 48	„besserer Zweig“, „optimale Route“, „ideale Linie“
22, 27	„schnellere[r] Zweig“, „schnellere Strecke“
31	„Die Wahl der schnellen oder breiten Strecke wird ggf. aufgrund des Fahrverhaltens des Teams gewählt.“
43	„Entscheidung für Richtung erfolgt je nach dem ob man sich weiter links oder rechts vor Gabelung befindet“
47	„In Abhängigkeit der Qualität der Fahrer, soll der kürzere, schwerere oder längere, einfachere Weg gewählt werden.“
38	„Zweig zufällig wählen“
24, 25, 28, 29, 30, 34, 36, 40, 41	Undefiniert

Tabelle 4.4: Variationen bei der Empfehlung des Gabelungszweigs

über diese machen. Z. B. möchte Team 23 den Fahrbahnrand hervorheben und schreibt „Je näher dem Rand, desto 'intensiver' warnen", macht aber keine genaueren Angaben dazu. Die Teams 39 und 45 schreiben in die allgemeine Automatikbeschreibung, dass sie Hinweise verwenden wollen, aber in den Funktionsbeschreibung wird das nicht weiter aufgegriffen.

- Analog verhält es sich auch mit auditiven Hinweisen (z. B. Teams 23, 29 - „Akkustik", 34, 39, 47 - „audio").
- Andere Teams (z. B. 25 und 31) sind präziser in ihren Beschreibungen und wollen zum Beispiel einen Pfeil anzeigen lassen. Aber auch hier ist unklar, wie groß dieser sein soll, welche Form er haben soll und wo er positioniert werden soll.

Bei vielen Funktionen ist auch nicht ersichtlich, wann genau diese aktiv sein sollen. Bei 35 von 68 hier behandelten Funktionen trifft das zu.

4.6 ZUSAMMENFASSUNG

Es werden die wesentlichen Erkenntnisse der Analyse zusammengefasst. Abbildung 4.1 visualisiert die Erkenntnisse als Mindmap.

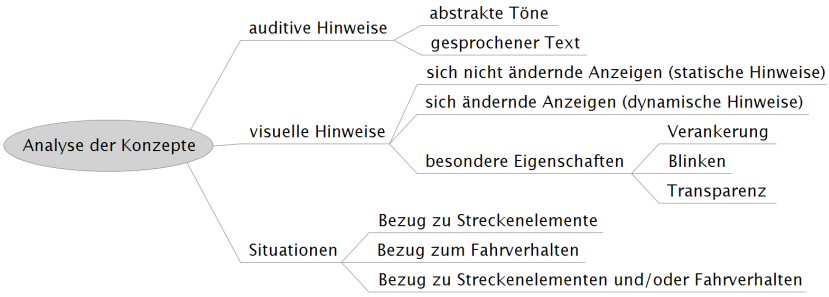


Abbildung 4.1: Mindmap zur Analyse der Konzeptbögen

Von den 30 Entwicklerteams haben sich alle für die Verwendung von auditiven oder visuellen Hinweisen ausgesprochen. Dies schlägt sich in den spezifizierten Funktionen der Konzepte nieder. Von den insgesamt 279 Funktionen sind 110 Funktionen den Hinweisen gewidmet. Mit dieser Arbeit werden 68 der 110 Funktionen umgesetzt.

Bedenkt man die kurze Entwicklungsphase von nur zwei Stunden, wird verständlich, dass sich in den meisten Konzepten Ungenauigkeiten wiederfinden.

Die meisten Hinweise werden eingesetzt, um die Konfliktsituation bei der Gabelung aufzulösen. Es haben fast alle Konzepte einen Hinweis für einen Richtungsvorschlag an Gabelungen vorgesehen. An zweiter Stelle folgen Hinweise, die die MWB an die

Geschwindigkeit erinnern. Auch sind Hinweise beliebt, die vor bestimmten Streckenelementen warnen, zum Beispiel vor Hindernissen.

Die auditiven Hinweise sind entweder den abstrakten Tönen wie einem „Bing“, „Piepton“, „Hupen“ oder einem gesprochenem Text, z. B. „Folgen Sie dem Pfeil!“, zuzuordnen.

Bei den visuellen Hinweisen kann zwischen statischen Hinweisen unterschieden werden, die ihre Anzeige nicht ändern, und dynamischen Hinweisen, die gerade das tun. Die dynamischen Hinweise sind oft so speziell, dass sie nur bei wenigen oder gar keinem anderen Konzept eingesetzt werden können. Einige Teams stellen besondere Eigenschaften an die visuellen Hinweise. Zum Beispiel ist bei einigen Hinweisen Transparenz, Blinken oder die Verankerung auf der Strecke gewünscht.

Die Hinweise werden in bestimmten Situationen ausgelöst. Es wird zwischen Situationen unterschieden, die auf bestimmten Streckenelementen basieren (Gabelungen, statische oder dynamische Hindernisse, Kurven verschiedenster Krümmung) und denen, die mit dem Fahrverhalten der MWB zusammenhängen (z. B. Lenkung auf Kollisionskurs, zu langsames oder zu schnelles Fahren). Einige Hinweise werden durch Situationen verschiedener Kategorien ausgelöst. Zum Beispiel kann ein Kollisionskurs nur in der Nähe des Streckenabschnitts festgestellt werden (Situation mit Bezug zum Fahrverhalten), in dem auch statische oder dynamische Hindernisse vorhanden sind (Situation mit Bezug zum Streckenelement).

5 ANFORDERUNGEN

Aufbauend auf der Analyse (Kapitel 4) wird in diesem Kapitel die benötigte Funktionalität zur Implementierung der auditiven und visuellen Hinweise festgelegt.

Dazu wird auf den Umgang mit ungenauen oder fehlenden Informationen in den Konzeptbögen eingegangen. Es wird der Bedarf an einem verallgemeinerten Ereignis-System erklärt sowie die Einführung des dazu notwendigen Ereignis-Modells vorgenommen. Danach werden alle erforderlichen Ereignisse und Hinweisfunktionen mit ihren Parametern spezifiziert. Schließlich werden die zu erbringende Arbeit in den Muss- und weitere vorteilhafte Funktionalität in den Wunschkriterien zusammengefasst.

5.1 UMGANG MIT FEHLENDEN INFORMATIONEN

Wie in Abschnitt 4.5 gezeigt, gibt es zahlreiche Konzepte, die Hinweise vorgesehen haben, diese aber nur vage beschreiben. Die folgenden Richtlinien beschreiben den Umgang mit diesen Hinweisen bei der Umsetzung. Die Richtlinien werden in gegebener Reihenfolge angewendet.

1. *Zusammenfassung und Parametrisierung von Hinweisen und Situationen*
   Diese Richtlinie bündelt die beschriebenen Hinweise und Situationen nach Ähnlichkeit und versieht diese mit Parametern, wodurch Übersicht gewonnen wird. Sollte sich die Interpretation einer Hinweisbeschreibung ändern, so ist die Wahrscheinlichkeit höher, dass die benötigte Funktionalität durch die nachträgliche Veränderung der Parameter umgesetzt werden kann (ohne Entwicklungsarbeit am Quelltext).

2. *Orientierung an ähnlichen Hinweisen anderer Konzepte*
   Diese Richtlinie erlaubt eine einigermaßen sinnvolle Umsetzung ungenauer Hinweise.

3. *Implementierung auf möglichst einfache Weise*
   Diese Richtlinie hält den Implementierungsaufwand für Hinweise mit sehr ungenauen und seltenen Anforderungen gering.

Der Richtlinie 1 nach werden in Abschnitt 5.4 die Situationen als Ereignisse zusammengefasst und parametrisiert. In Abschnitt

5.5 geschieht das Gleiche für die Hinweise - diese werden als „Hinweisfunktionen“ umgesetzt.

Für die weitere Anwendung der Richtlinien sei auf Kapitel 6 und insbesondere Anhang A verwiesen.

5.2 BEDARF AN EINEM PARAMETRISIERTEN EREIGNIS-SYSTEM

Abschnitt 4.3.3 im Kapitel „Analyse“ hat gezeigt, dass für die Hinweise verschiedene Situationen zur Aktivierung erkannt werden müssen. Wird beispielsweise erkannt, dass die MWB das Objekt zu schnell steuern, kann ein Hinweis eingeblendet werden, der sie darauf aufmerksam macht. Die Situation „zu schnell fahren“ kann aber auch für andere Funktionen in dem Automatiken-GUI nützlich sein. Zum Beispiel könnte eine Funktion einer Steuerungsautomatik bei der Situation aktiv werden und die MWB in ihrer vertikalen Joystick-Auslenkung einschränken, um die Geschwindigkeit zu drosseln. Tatsächlich können viele der hier genannten Situationen in anderen Funktionen wiederverwendet werden. Daher ist es sinnvoll, die Erkennung von Situationen auch für andere Funktionen zur Verfügung zu stellen.

Im Ansatz hat Fuhrmann (2010) diese Idee bereits in dem von ihr entwickelten Automatiken-GUI umgesetzt. Abbildung 5.1 zeigt einen Ausschnitt aus dem Eigenschaftenbereich des Automatiken-GUI. Die Funktion „Lenken“ ist angeklickt. Der Benutzer kann für diese Funktion mehrere „aktive Streckentypen“ auswählen. Dazu gehören Gerade, Kurve und Gabelung. Dies entspricht Situationen, die auf Streckenelementen basieren (vgl. Abschnitt 4.3.3).

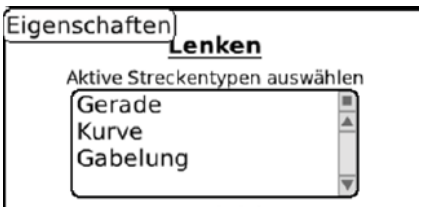


Abbildung 5.1: Für eine Funktion innerhalb des Automatiken-GUI können „aktive Streckentypen“ festgelegt werden.

Für die Implementierung der hier vorgestellten auditiven und visuellen Hinweisen müssen auch Situationen, die auf dem Fahrverhalten der MWB basieren, hinzugefügt werden. Weiterhin müssen einige Situationen konfigurierbar sein. So wäre es zur Zeit nicht möglich, Hinweise mit Warncharakter umzusetzen, da diese Funktionen aktiviert werden müssen, bevor das Streckenelement vom Objekt passiert wird. Dazu muss der Benutzer z. B. angeben, wie viele Pixel vor einem Streckenelement die Situation erkannt werden soll. Ein anderes Beispiel ist die bereits oben erwähnte Situationen „zu schnell fahren“. Hier wäre es sinnvoll, wenn der

Benutzer eine Toleranz angeben könnte, damit der entsprechende Hinweis nicht ständig erscheint, sobald die MWB nur minimal die optimale Geschwindigkeit überschreiten.

5.3 DAS EREIGNIS-MODELL

An dieser Stelle wird das benötigte Ereignis-Modell beschrieben, um die Hinweise zu den festgelegten Zeitpunkten anzuzeigen bzw. abzuspielen.

Ein Ereignis ist ein boolescher Ausdruck, der zu jedem Simulationsschritt evaluiert werden kann. Das *Ereignis tritt ein*, wenn der Ausdruck zu wahr evaluiert. Anderenfalls tritt das *Ereignis nicht ein*.

Es wird zwischen zwei Typen von Ereignissen unterschieden:

- *Ereignis mit Bezug zu einem Streckenelement*
  Tritt ein, sobald ein Streckenelement sich nähert oder vom Objekt passiert wird.
- *Ereignis mit Bezug zu dem Fahrverhalten der* MWB
  Tritt ein, sobald ein bestimmtes Verhalten der MWB festgestellt wird.

Ein eintretendes Ereignis *aktiviert* die zugehörige Funktion, d. h. der visuelle Hinweis wird angezeigt bzw. der auditive Hinweis wird abgespielt. Durch Verknüpfung der Ereignisse mit den logischen UND-, ODER- und NEGATIONs-Operator können recht präzise Ereignisse definiert werden.

Ereignisse können durch *Parameter* konfiguriert werden. Diese hängen vom konkreten Ereignis ab.

Alle *Ereignisse mit Bezug zu einem Streckenelement* verfügen über die zwei Standardparameter *untere* und *obere Schranke*. Die untere und obere Schranke wird relativ zum Beginn des Streckenelements angegeben. In Abbildung 5.2 ist eine Gabelung schematisch dargestellt[1]. Der Beginn dieses Streckenelements ist in der Abbildung durch die rote, horizontale Linie markiert. Die untere Schranke kann z. B. durch *Gabelungsbeginn - 150* und die obere Schranke durch *Gabelungsbeginn + 200* angegeben werden. Wie der Name bereits verrät, gilt stets *untere Schranke* < *obere Schranke*. Die beiden Schranken definieren einen *Aktivierungsbereich*. Befindet sich das Objekt in[2] diesem Bereich, so tritt das zugehörige Strecken-Ereignis ein und die Funktion wird aktiv. Durch *obere Schranke* < *Gabelungsbeginn* kann der Aktivierungsbereich vor dem Gabelungsbeginn enden (analog für danach). Dadurch ist es möglich, Funktionen vor dem zu warnenden Streckenelement zu aktivieren.

1 Sei der Koordinatenursprung links unten angenommen.
2 Es soll verglichen werden, ob die oberste Pixelzeile des Objekts in dem Bereich liegt. Die Grenzen des Aktivierungsbereichs sind mit eingeschlossen.

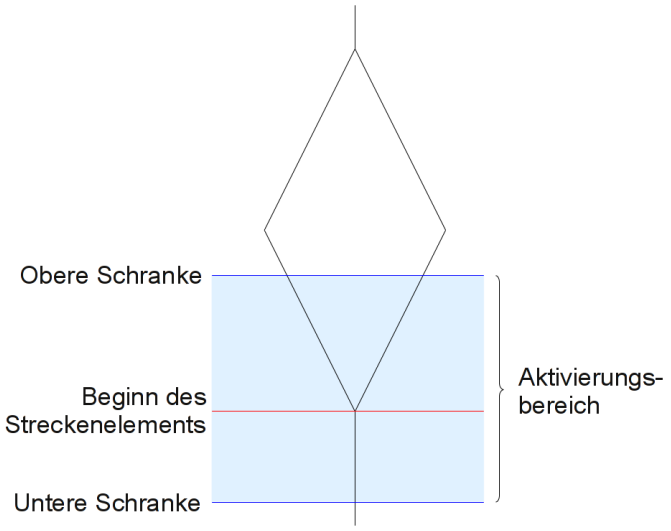


Abbildung 5.2: Beispiel für einen Aktivierungsbereich eines Ereignisses für Gabelungen.

Für Ereignisse mit Bezug zu dem Fahrverhalten der MWB ist es nicht vorhersehbar, wann die Ereignisse eintreten und wie lange diese andauern. Daher sind für diese Ereignisse Schranken nicht sinnvoll.

Mit diesem Ereignis-Modell können die in Abschnitt 4.3.3 aufgeführten Situationen implementiert werden. Ein Ereignis kann mehrere Situationen realisieren. Eine konkrete Situation kann durch die Parameter des Ereignisses konfiguriert werden.

5.4 ERFORDERLICHE EREIGNISSE

Im Folgenden werden die notwendigen Ereignisse für die Umsetzung der Hinweise spezifiziert. Ein Ereignis wird durch seinen Namen und folgende Punkte beschrieben:

VORAUSSETZUNGEN sind die Bedingungen, die gelten müssen, damit das Ereignis verwendet werden kann.

EREIGNISTYP ist entweder „Strecke“ oder „Fahrverhalten“.

PARAMETER dienen der Konfiguration des Ereignisses. Bei dem Ereignistyp „Streckenbezogen“ sind die Parameter „Vor- und Nachlaufschranke“ immer vorhanden.

EINTRITT informiert darüber, wann das Ereignis eintritt und somit den assoziierten Hinweis aktiviert.

ANMERKUNGEN enthält weiterführende Beschreibungen, z. B. zu den Parametern oder dem Eintritt des Ereignisses.

Mit diesem Ereignis-Modell können die in Abschnitt 4.3.3 eingeführten „Situationen“ implementiert werden.

5.4.1 *Ereignisse mit Bezug zu Streckenelementen*

Es werden die Ereignisse mit Bezug zu Streckenelementen spezifiziert. Der Ereignistyp ist bei diesen daher implizit auf „Strecke“ festgelegt.

EREIGNIS *„Bevor/im Gabelungsbereich“*

Voraussetzungen:
(1) Position des Objekts bekannt
(2) Position der Gabelung bekannt
(3) Gabelungstypen unterscheidbar
Parameter:
(1) *Untere Schranke* (rel. zur untersten Pixelzeile der Gabelungskachel)
(2) *Obere Schranke* (rel. zur untersten Pixelzeile der Gabelungskachel)
(3) *Gabelungstyp*:
„beliebige Gabelung“,
„runde Gabelung“,
„eckige Gabelung“
Eintritt:
Oberste Pixelzeile des Objekts ist im Aktivierungsbereich.
Anmerkungen:

Dieses Ereignis erkennt Gabelungen auf der Strecke. Durch Angabe von unterer und oberer Schranke kann der Aktivierungsbereich konfiguriert werden.

Die Unterscheidung nach Typen ist notwendig, da z. B. in Konzept 37 der breitere Gabelungszweig vorgeschlagen wird.

EREIGNIS *„Bevor/im Hindernisbereich“*

Voraussetzungen:
(1) Position des Objekts bekannt
(2) Position des Hindernisses bekannt
(3) Hindernistypen unterscheidbar
Parameter:
(1) *Untere Schranke* (rel. zur untersten Pixelzeile des Hindernisses)
(2) *Obere Schranke* (rel. zur untersten Pixelzeile des Hindernisses)
(3) *Hindernistyp*:
„beliebig“
„dynamisches Hindernis“
„statisches Hindernis, 25% Abdeckung linke Fahrbahn“
„statisches Hindernis, 50% Abdeckung linke Fahrbahn“
„statisches Hindernis, 25% Abdeckung rechte Fahrbahn“
„statisches Hindernis, 50% Abdeckung rechte Fahrbahn“
Eintritt:
Oberste Pixelzeile des Objekts ist im Aktivierungsbereich.

Anmerkungen:

Dieses Ereignis erkennt Hindernisse auf der Strecke. Durch Angabe von unterer und oberer Schranke kann der Aktivierungsbereich konfiguriert werden.

Die Unterscheidung nach Typen ist notwendig, da sonst nicht festgelegt werden kann, wie die Route zum Umfahren statischer Hindernisse (z. B. bei Konzept 25) gelegt werden soll.

Durch dieses Ereignis werden die Situationen „Statische Hindernisse“ und „Dynamische Hindernisse“ aus Abschnitt 4.3.3 zusammengefasst. Durch den *Hindernistyp* kann zwischen diesen beiden unterschieden werden.

EREIGNIS *„Bevor/im Kurvenbereich“*

Voraussetzungen:
(1) Position des Objekts bekannt
(2) Position der Kurven bekannt
(3) Kurvenkrümmungen unterscheidbar
Parameter:
(1) *Untere Schranke* (rel. zur untersten Pixelzeile der Kurve)
(2) *Obere Schranke* (rel. zur untersten Pixelzeile der Kurve)
(3) *Kurvenkrümmungen*: Bei der Implementierung zu ermitteln.
Eintritt:
Oberste Pixelzeile des Objekts ist im Aktivierungsbereich.
Anmerkungen:

Dieses Ereignis erkennt Kurven auf der Strecke. Durch Angabe von unterer und oberer Schranke kann der Aktivierungsbereich konfiguriert werden.

Die Unterscheidung nach *Kurvenkrümmungen* ist notwendig, damit vor „scharfen Kurven“ gewarnt werden kann (gewünscht von Team 46). In SAM sind keine Informationen zu Kurven enthalten. Die unterschiedlichen *Kurvenkrümmungen* sind noch zu ermitteln.

5.4.2 *Ereignisse mit Bezug zum Fahrverhalten*

Im Folgenden werden die Ereignisse mit Bezug zum Fahrverhalten spezifiziert. Der Ereignistyp ist bei diesen daher implizit auf „Fahrverhalten“ festgelegt.

EREIGNIS *„Auf der Fahrbahn“*

Voraussetzungen:
(1) Objekt-Sensoren gegeben
Parameter:
(1) *Sensorenanzahl* (1 - 8)
Eintritt:

Es sind mindestens so viele Sensoren des Objekts auf der Fahrbahn, wie durch *Sensorenanzahl* angegeben.
Anmerkungen: keine

EREIGNIS *„Kollisionskurs mit Hindernissen“*

Voraussetzungen:
(1) Position des Objekts bekannt
(2) Ereignis aktiv: „Hindernis“
(3) Joystick-Eingaben gegeben
Parameter: keine
Eintritt:
Beibehaltung der aktuellen Joystick-Eingaben führt zu Kollision des Objekts mit Hindernis.
Anmerkungen:

Für dynamische Hindernisse wird deren Geschwindigkeit berücksichtigt.

Angenommen dem Objekt sind zwei statische Hindernisse voraus. Hindernis A liegt vor Hindernis B und ist auf der linken Fahrspur. Hindernis B liegt auf der rechten Fahrspur. Wenn das Objekt auf der linken Fahrspur Richtung Hindernis A gelenkt wird, dann befindet es sich auf Kollisionskurs. Wird das Objekt dann auf die rechte Fahrspur manövriert, um Hindernis A auszuweichen, dann befindet es sich immer noch auf Kollisionskurs, da die Lenkung dann zumindest kurzzeitig auf Hindernis B abzielt. Das ist ein korrektes Verhalten.

EREIGNIS *„Geschwindigkeitsempfehlung“*

Voraussetzungen:
(1) Aktuelle Geschwindigkeit des Objekts bekannt
(2) Optimale Geschwindigkeit des Objekts bekannt
(3) Ereignis aktiv: „Auf der Fahrbahn“ mit Sensorenanzahl = 1
(4) Ereignis aktiv: „Kollisionskurs mit Hindernissen“
Parameter:
(1) *Empfehlungstyp*:
„Schneller fahren“
„Langsamer fahren“
(2) *Ausschließlich dynamische Hindernisse berücksichtigen*:
„Ja“
„Nein“
(3) *Geduldete Abweichung* (in Pixel/Tick)
Eintritt:

Seien $v_{aktuell}$ die aktuelle Geschwindigkeit des Objekts, $v_{optimal}$ die optimale Geschwindigkeit auf der Mittellinie und d die *geduldete Abweichung* ist.

1. *Empfehlungstyp* „Langsamer fahren“:

a) *Ausschließlich dynamische Hindernisse berücksichtigen* „Ja“: Bei Kollisionskurs mit dynamischen Hindernis: Bremsen vermeidet Kollision

b) *Ausschließlich dynamische Hindernisse berücksichtigen* „Nein“:

i. Falls Kollisionskurs mit dynamischen Hindernis: Bremsen vermeidet Kollision

ii. Sonst: $v_{aktuell} > v_{optimal} + d$

2. *Empfehlungstyp* „Schneller fahren“:

a) *Ausschließlich dynamische Hindernisse berücksichtigen* „Ja“: Bei Kollisionskurs mit dynamischen Hindernis: Beschleunigen vermeidet Kollision

b) *Ausschließlich dynamische Hindernisse berücksichtigen* „Nein“:

i. Falls Kollisionskurs mit dynamischen Hindernis: Beschleunigen vermeidet Kollision

ii. Sonst: $v_{aktuell} < v_{optimal} - d$

In den Fällen, in denen $v_{aktuell}$ und $v_{optimal}$ berücksichtigt werden, wird für $d = 0$ das Ereignis immer eintreten, sobald das Objekt sich langsamer oder schneller als optimal bewegt.
Anmerkungen:

Durch den *Empfehlungstyp* kann festgelegt werden, dass das Ereignis eintritt, wenn das Objekt entweder zu langsam oder zu schnell fährt. Der Geschwindigkeitsvergleich erfolgt mit der aktuellen Geschwindigkeit des Objekts und einer vorher ermittelten optimalen Geschwindigkeit für die Strecke. Die optimale Geschwindigkeit ist die höchste Geschwindigkeit, die vom Objekt eingehalten werden kann, ohne von der Mittellinie der Fahrbahn abzukommen. Diese Definition unterstützt beide Ziele der MWB, in dem der Flächenfehler gering gehalten wird (auf der Mittellinie fahren) und die höchste mögliche Geschwindigkeit gefahren wird (möglichst schnell fahren). Für ein Konzept, das sowohl einen Geschwindigkeitshinweis bei zu schnellem als auch bei zu langsamem Fahren einblenden möchte, kann mit der *geduldeten Abweichung* ein Toleranzbereich definiert werden.

Wenn das Objekt nicht auf der Fahrbahn ist, tritt das Ereignis nicht ein, da der Vergleich mit der optimalen Geschwindigkeit nur sinnvoll ist, wenn das Objekt sich zumindest annähernd auf der Fahrbahn befindet.

Dynamische Hindernisse sind ein Spezialfall. Sobald ein Kollisionskurs mit einem dynamischen Hindernis festgestellt wird, findet kein Abgleich mit der ermittelten optimalen Geschwindigkeit statt. Stattdessen wird ermittelt, ob das Objekt so beschleunigt bzw. abgebremst werden kann, dass es ohne Kollision den Streckenabschnitt passiert. Die *geduldete Abweichung* hat keine Relevanz bei Kollisionskurs mit dynamischen Hindernissen.

Der Parameter *Ausschließlich dynamische Hindernisse berücksichtigen* ist für Konzepte vorgesehen, die nur eine Geschwindigkeitsempfehlung bei dynamischen Hindernissen benötigen.

EREIGNIS *„Abstand zum Fahrbahnrand unterschritten“*

Voraussetzungen:
(1) Position des Objekts bekannt
(2) Ereignis aktiv: „Auf der Fahrbahn“ mit Sensorenanzahl = 8
Parameter:
(1) *Geduldeter Abstand*
Eintritt: *Geduldeter Abstand* zwischen äußerem Objektrand und Fahrbahnrand ist unterschritten.
Anmerkungen: keine

EREIGNIS *„Differenz der Joystick-Eingaben“*

Voraussetzungen:
(1) Joystick-Eingaben gegeben
Parameter:
(1) *Geduldeter Abweichungswinkel* (zw. den Richtungsvektoren)
Eintritt:
Der Winkel zwischen den beiden Richtungsvektoren ist größer als der *geduldete Abweichungswinkel*.
Anmerkungen: keine

EREIGNIS *„Lenkung in gewünschte Richtung“*

Voraussetzungen:
(1) Joystick-Eingaben gegeben
Parameter:
(1) *Geduldeter Abweichungswinkel*
(2) *Zielpunkt* (definiert gewünschte Richtung)
Eintritt:
Die aktuelle Richtung des Objektes entspricht, wenn der *geduldete Abweichungswinkel* miteinbezogen wird, dem des *Zielpunkts*.
Anmerkungen: keine

EREIGNIS *„Personenausfall“*

Voraussetzungen:
(1) Joystick-Eingaben gegeben
Parameter:
(1) *Eingabe-Timeout* (in ms)
(2) *Timeout-Betroffene*:
„Mikroweltbewohner 1“
„Mikroweltbewohner 2“
„Beide Mikroweltbewohner“
Eintritt:

Der oder die *Timeout-Betroffene* lässt bzw. lassen den Joystick für mehr als *Eingabe-Timeout* ms in Ruhestellung.
Anmerkungen: keine

EREIGNIS *„Ungenaues Fahren“*

Voraussetzungen:
(1) Grad für ungenaues Fahren bekannt
Parameter:
(1) *Schwellenwert*
Eintritt:
Der *Schwellenwert* für ungenaues Fahren wird überschritten.
Anmerkungen: „Ungenaues Fahren“ ist noch zu definieren und sollte möglichst einfach umgesetzt werden.

EREIGNIS *„Links oder rechts von der Mittellinie“*

Voraussetzungen:
(1) Joystick-Eingaben gegeben
(2) Objekt-Sensoren gegeben
Parameter:
(1) *Seite*:
„links“
„rechts“
Eintritt:
Es wird auf der gewünschten Seite der Mittellinie gefahren.
Anmerkungen:

Diese Spezifikation ist eine Interpretation von Konzept 43, in dem es heißt „je nachdem auf welcher Fahrspur sich das Objekt befindet, wird die optimale Route gewählt und angezeigt“. Eine Beschränkung auf die linke oder rechte Fahrspur wird hier nicht vorgenommen, da es passieren kann, dass das Objekt abseits der Fahrbahn fährt. Für diesen Fall ist es für Konzept 43 sinnvoll zu wissen, ob das Objekt links oder rechts der Mittellinie ist. Eine Einschränkung auf die Fahrbahn würde eine Entscheidung in diesem Fall nicht möglich machen.

Die Mittellinie wird als der gewünschten Seite zugehörig betrachtet.

5.4.3 *Ereignisse und Abhängigkeiten untereinander*

Wie aus der Spezifikation der Ereignisse entnommen werden kann, setzen einige Ereignisse andere Ereignisse voraus. Abbildung 5.3 fasst alle Ereignisse und ihre Abhängigkeiten untereinander zusammen.

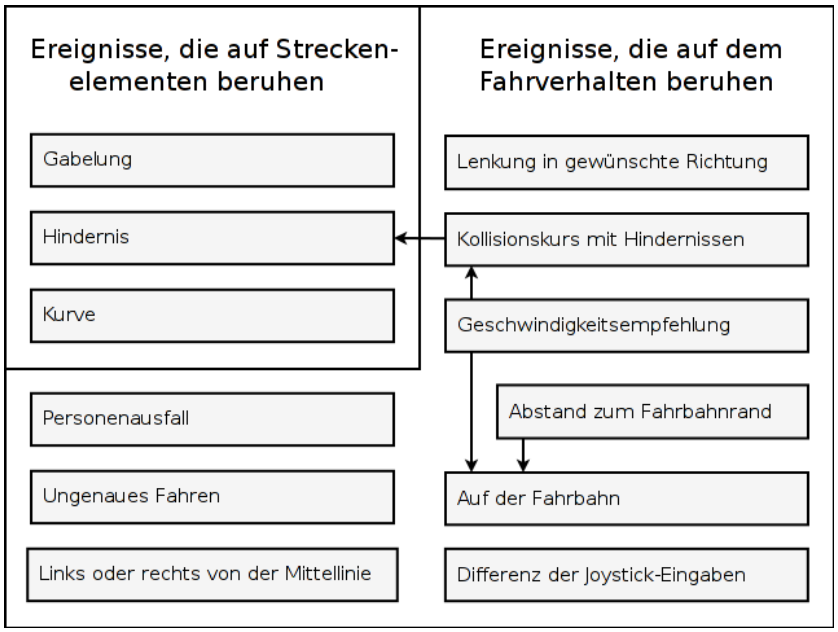


Abbildung 5.3: Ereignisse und ihre Abhängigkeiten untereinander. „A → B“ steht für „A benutzt B“.

5.5 ERFORDERLICHE HINWEISFUNKTIONEN

Im Kontext des Automatiken-GUI werden sogenannte „Funktionen“ konfiguriert. In diesem Abschnitt werden entsprechend „Hinweisfunktionen“ spezifiziert, durch deren geeignete Konfiguration in dem Automatiken-GUI die Hinweise aus den Konzepten umgesetzt werden können.

5.5.1 *Hinweisfunktion „Hinweis (auditiv)“*

Die in Abschnitt 4.3.1 vorgestellten abstrakten Töne sowie der gesprochene Text können als Audiodatei abgespeichert werden. Dadurch ergibt sich nur eine Hinweisfunktion, mit der alle auditiven Hinweise umgesetzt werden können:

Hinweisfunktion „Hinweis (auditiv)“
Parameter:
(1) *Pfad zur Audiodatei*

Der Piepton, der seine Frequenz abhängig vom Abstand zu einem Streckenelement ändert (Konzept 24), kann mit dieser Hinweisfunktion ebenfalls umgesetzt werden. Dazu können durch die streckenbezogenen Ereignisse verschiedene Abstände definiert werden. Je nach Abstand wird ein Ton mit einer anderen Frequenz abgespielt. Die Frequenz wird also stufenweise angepasst, nicht fließend (vgl. Richtlinie 3 in Abschnitt 5.1).

Durch den Parameter ist diese Hinweisfunktion generisch genug, um auch zukünftige auditive Hinweise ohne Entwicklungsarbeit am Quelltext umzusetzen.

5.5.2 *Hinweisfunktion „Hinweis (visuell)“*

Wie alle auditiven Hinweise, können alle statischen visuellen Hinweise mit nur einer Hinweisfunktion umgesetzt werden. Der Parameter ist der unveränderte visuelle Hinweis als Bilddatei. In dieser sind die Details des gewünschten Hinweises abgebildet - Symbole, Piktogramme oder Schriftzüge. Neben dem Pfad zur Bilddatei als Parameter müssen auch die in Abschnitt 4.3.2 aufgezeigten besonderen Eigenschaften von visuellen Hinweisen konfigurierbar sein.

Hinweisfunktion „Hinweis (visuell)“
Parameter:
(1) *Pfad zur Bilddatei*
(2) *Blinken*: „Ja“, „Nein“
(2.1) *Einblendedauer*: 0 - 10000 in ms
(2.2) *Ausblendedauer*: 0 - 10000 in ms
(3) *Einblenden mit Verankerung*: „Ja“, „Nein“
(4) *X-Koordinate*: 0 - 768.
(5) *Y-Koordinate*: 0 - 800.
(6) *Transparenz*: 5 - 95 als %

Durch den Parameter *Pfad zur Bilddatei* ist diese Hinweisfunktion generisch genug, um auch zukünftige visuelle Hinweise ohne Implementierungsaufwand umzusetzen. Insbesondere ist die Auslagerung in die Bilddatei für zukünftige Hinweise interessant, die bestimmte Schriftarten voraussetzen, die nicht in Squeak verfügbar sind.

Ist *Blinken* auf „Ja“ gesetzt, dann kann über die *Einblende-* und *Ausblendedauer* festgelegt werden, wie lange das Bild angezeigt werden soll und wie lange die Pause dauert, bevor das Bild wieder erneut eingeblendet wird. Ist *Blinken* auf „Nein“ gesetzt, wird das Bild so lange angezeigt, wie das zugehörige Ereignis eintritt.

Einblenden mit Verankerung ist in Abschnitt 4.3.2 beschrieben. Ein „Nein“ entspricht Einblendung ohne Verankerung. Die *X-* und *Y-Koordinaten* richten sich nach diesem Parameter.

Die Parameter *Transparenz* dient der Optimierung der Anzeige. Ohne *Transparenz* könnten Hinweise die Strecke, insbesondere auch die Fahrbahn in einigen Kurven, vollständig über mehrere Simulationsschritte verdecken. Die *Transparenz* kann in Prozent angegeben werden. Bei weniger als 5% wird die Transparenz

kaum wahrgenommen und bei mehr als 95% wird das Bild kaum erkennbar.

5.5.3 *Hinweisfunktion „Gabelungshinweis (visuell)“*

Mit der folgenden Hinweisfunktion werden die visuellen Richtungsvorschläge bei Gabelungen abgedeckt.

Hinweisfunktion „Richtungsvorschlag bei Gabelung“
Parameter:
(1) *Pfad zur Bilddatei für linken Zweig (runde Gabelung)*
(2) *Pfad zur Bilddatei für rechten Zweig (runde Gabelung)*
(3) *Pfad zur Bilddatei für linken Zweig (eckige Gabelung)*
(4) *Pfad zur Bilddatei für rechten Zweig (eckige Gabelung)*
(5) *Richtungsvorschlag*:
„Besserer/Optimaler Zweig“
„Dünner Zweig“
„Breiter Zweig“
„Kurzer Zweig“
„Langer Zweig“
„Schneller Zweig“
„Rechter Zweig“
„Einfacher Zweig“
„Schwieriger Zweig“
„Zufälliger Zweig“
(6) *Blinken*: „Ja“, „Nein“
(6.1) *Einblendedauer*: 0 - 10000 in ms
(6.2) *Ausblendedauer*: 0 - 10000 in ms
(7) *Einblenden mit Verankerung*: „Ja“, „Nein“
(8) *X-Koordinate*: 0 - 768.
(9) *Y-Koordinate*: 0 - 800.
(10) *Transparenz*: 5 - 95 in %

Diese Hinweisfunktion entspricht im Wesentlichen der generischen Hinweisfunktion „visueller Hinweis“ für statische visuelle Hinweise. Der Unterschied liegt in den ersten drei Parametern. Zum einen müssen vier *Bilddateien* angegeben werden, da sonst nicht präzise Pfade für die Zweige der Gabelungen realisiert werden können. Zum anderen muss der *Richtungsvorschlag* festgelegt werden.

5.5.4 *Hinweisfunktion „Gabelungshinweis (auditiv)“*

Folgende Hinweisfunktion deckt alle auditiven Richtungsvorschläge bei Gabelungen ab.

Hinweisfunktion „Gabelungshinweis (auditiv)“

Parameter:
(1) *Pfad zur Audiodatei für linken Zweig*
(2) *Pfad zur Audiodatei für rechten Zweig*
(3) *Richtungsvorschlag*:
„Besserer/Optimaler Zweig“
„Dünner Zweig“
„Breiter Zweig“
„Kurzer Zweig“
„Langer Zweig“
„Schneller Zweig“
„Rechter Zweig“
„Einfacher Zweig“
„Schwieriger Zweig“
„Zufälliger Zweig“

Auch diese Hinweisfunktion entspricht im Wesentlichen der generischen Hinweisfunktion „auditiver Hinweis“. Für die beiden Gabelungszweige wird jeweils eine *Audiodatei* benötigt. Der *Richtungsvorschlag* ist ebenso notwendig.

5.5.5 *Hinweisfunktion „Geschwindigkeitshinweis“*

Die Empfehlung, die Geschwindigkeit anzupassen, ist ebenfalls oft erwünscht und soll mit folgender Hinweisfunktion umgesetzt werden:

Hinweisfunktion „Geschwindigkeitshinweis“
Parameter:
(1) *Pfad zur Bilddatei für Geschwindigkeitserhöhung*
(2) *Pfad zur Bilddatei für Geschwindigkeitserniedrigung*
(3) *Blinken*: „Ja“, „Nein“
(3.1) *Einblendedauer*: 0 - 10000 in ms
(3.2) *Ausblendedauer*: 0 - 10000 in ms
(4) *Einblenden mit Verankerung*: „Ja“, „Nein“
(5) *X-Koordinate*: 0 - 768.
(6) *Y-Koordinate*: 0 - 800.
(7) *Transparenz*: 5 - 95 in %

Diese Hinweisfunktion entspricht im Wesentlichen auch der generischen Hinweisfunktion „visueller Hinweis“ für statische visuelle Hinweise. Durch die ersten beiden Parameter werden die beiden Bilddateien zur Empfehlung festgelegt.

5.5.6 *Hinweisfunktion „Streckenvorschau“*

Die Streckenvorschau wird in den Konzepten 37 und 44 beschrieben. Konzept 37 nach soll auch der Zoomfaktor der Streckenvor-

schau der Geschwindigkeit angepasst werden. Je schneller das Objekt fährt, desto weiter sollten die MWB vorausschauen können.

Hinweisfunktion „Streckenvorschau“
Parameter:
(1) *X-Koordinate*: 0 - 768
(2) *Y-Koordinate*: 0 - 800
(3) *Breite*: 5 - 30 in % von 768
(4) *Höhe*: 5 - 100 in % von 800
(5) *Vorschau der Geschwindigkeit anpassen*: „Ja“, „Nein“
(6) *Transparenz*: 5 - 95 in %

Die ersten vier Parameter decken die Positionierung und Größe der Streckenvorschau ab.

Ist Parameter *Vorschau der Geschwindigkeit anpassen* auf „Ja“ eingestellt, wird der Zoomfaktor entsprechend der Geschwindigkeit verändert. Dabei verändert sich auch die Positionierung und Größe der Karte.

5.6 ZUSAMMENFASSUNG

In diesem Kapitel wurden die Anforderungen für die Umsetzung der auditiven und visuellen Hinweise erarbeitet.

Den teilweise vagen Beschreibungen in den Konzepten wird in erster Linie durch Konfigurationsmöglichkeiten begegnet. Anstatt jeden einzelnen Hinweis isoliert von den anderen zu implementieren, werden die Hinweise durch „Hinweisfunktionen“ umgesetzt. Diese fassen die benötigte Funktionalität ähnlicher Hinweise zusammen. Die Einstellung der sich ergebenden Parameter ermöglicht die Umsetzung eines bestimmten Hinweises durch Konfiguration der passenden Hinweisfunktion. Die Anzahl der Hinweisfunktionen ist wesentlich geringer als die Anzahl der Hinweise, was die Übersicht erhöht.

Analog wurden die in der Analyse festgestellten Situationen als Ereignisse umgesetzt werden. Dazu wurde ein passendes Ereignis-Modell definiert, welches vorsieht, die Hinweisfunktionen noch vor bestimmten Streckenelementen aktivieren zu können, um Hinweise mit Warncharakter zu implementieren. Im Modell verfügen die Ereignisse ebenfalls über Parameter, sodass mehrere Situationen als ein Ereignis zusammengefasst werden können.

Die erforderlichen Ereignisse und Hinweisfunktionen wurden schließlich spezifiziert. Zusammenfassend werden die Anforderungen im Folgenden als Muss- und Wunschkriterien aufgeführt.

5.6.1 *Musskriterien*

Die folgenden Anforderungen müssen für die Implementierung der auditiven und visuellen Hinweise umgesetzt werden.

1. Alle Ereignisse aus Abschnitt 5.4.1 und 5.4.2 sollen für die Aktivierung von beliebigen Funktionen zur Verfügung stehen.
2. Ereignisse sollen miteinander durch den UND-, ODER- und NICHT-Operator verknüpfbar sein.
3. Alle Hinweisfunktionen im Abschnitt 5.5 sollen umgesetzt werden.
4. Alle Hinweisfunktionen sollen über das Automatiken-GUI konfiguriert werden können.

5.6.2 *Wunschkriterien*

Die folgenden Anforderungen sind nicht unbedingt erforderlich, aber sehr vorteilhaft.

1. Die Konfiguration der Hinweisfunktionen sollte nach softwareergonomischen Gesichtspunkten gestaltet sein.
2. Die Konfiguration der Ereignisse einer Automatik soll möglichst flexibel sein.

Während mit dem ersten Punkt die Benutzer viel Zeit sparen können, werden mit Punkt zwei zukünftige Ereignisse, insbesondere Ereignisse, die in den Konzeptbögen nicht beschrieben wurden, ermöglicht.

6 PRAKTISCHE UMSETZUNG

In diesem Kapitel wird aufbauend auf den Anforderungen des Kapitels 5 die Umsetzung des Ereignis-Modells (Abschnitt 6.1), der Ereignisse (Abschnitt 6.2) und der Hinweisfunktionen (Abschnitt 6.3) beschrieben. Schließlich wird in Abschnitt 6.4 erklärt, warum die entwickelten Benutzungsschnittstellen gut sind.

Die Dokumentation der Hinweisfunktionen und der Ereignisse befindet sich in der Bedienungsanleitung des Automatiken-GUI in den Abschnitten C.12 und C.13. Wie die Aktivierung einer Funktion durch Ereignisse konfiguriert werden kann, wird in Abschnitt C.4.7 beschrieben.

Die in diesem Kapitel beschriebene Umsetzung bildet die Grundlage für Implementierung der Funktionen aus den Konzeptbögen (vgl. Kapitel 4). Diese Funktionen werden durch Konfiguration der Hinweisfunktionen implementiert. Aufgrund der Fülle der Hinweisfunktionen konnten nicht alle umgesetzt werden. In Anhang A werden die implementierten Funktionen aufgelistet.

6.1 EREIGNIS-MODELL

Die Ereignisse sind als Klassen implementiert und die Parameter der Ereignisse werden als Instanzvariablen gespeichert. Alle Ereignisklassen werden direkt oder indirekt von der abstrakten Basisklasse `AAFAbstractEvent` abgeleitet. Über die Methode `isValid:` wird anhand des übergebenen SamStates bestimmt, ob das Ereignis für den aktuellen Simulationsschritt eintritt oder nicht.

```
AAFAbstractEvent isValid: aSamState
        self subclassResponsibility
```

Ereignisklassen müssen diese Methode sinnvoll implementieren. Die Methode gibt `true` zurück, falls das Ereignis eintritt, ansonsten `false`.

Die Ereignisse können miteinander verknüpft werden, da der UND- sowie ODER-Operator umgesetzt wurde. Beliebige UND/ODER-Ausdrücke wie z. B. $E_1 \vee (E_2 \wedge E_3)$ können somit konstruiert werden. Die Operanden E_1, E_2 und E_3 sind Ereignisse, welche erst zur Laufzeit, jeweils für einen Simulationsschritt, ausgewertet werden. Für den NICHT-Operator gab es keinen praktischen Bedarf, sodass dieser nicht implementiert wurde.

Ein Ausdruck wird intern in einer Baumstruktur repräsentiert. Die Blätter des Baums sind Ereignisse und die inneren Knoten sind UND/ODER-Operatoren. Der Ausdruck wird in drei verschiedenen Repräsentationen gespeichert, welche in den folgenden Abschnitten erklärt werden. Anschließend wird darauf eingegangen, wie das umgesetzte Ereignis-Modell mit den Funktionen verbunden wird.

6.1.1 *Repräsentation zur Laufzeit*

Jeder `AAFDelegate` (und damit jeder `AAFAgent` und `AAFGraph`) verfügt über eine Instanzvariable `event`, welche den Ausdruck repräsentiert. Die Instanzvariable `event` ist entweder ein Ereignisobjekt (Instanz einer Unterklasse von `AAFAbstractEvent`), falls es sich um keinen zusammengesetzten Ausdruck handelt, oder eine Instanz von der Klasse `AAFBoolOperator`. Diese Klasse repräsentiert entweder einen UND- oder ODER-Operator und verfügt wie `AAFAbstractEvent` ebenfalls über die Methode `isValid:`, welche den Ausdruck evaluiert. Durch diese einfache Schnittstelle muss der Aufrufer von `event isValid: samState` nicht um den dahinterstehenden Typen (`AAFAbstractEvent` oder `AAFBoolOperator`) wissen. Ein ausgezeichneter Wurzelknoten(typ) wird nicht benötigt.

Abbildung 6.1 veranschaulicht den Ausdruck $E_1 \vee (E_2 \wedge E_3)$ von weiter oben in dieser Repräsentation. Die Rechtecke stellen Objekte dar und die Linien Referenzen zwischen diesen. Der konkrete Operator einer `AAFBoolOperator`-Instanz wird über die Instanzvariable `type` festgelegt. `AAFBoolOperator` verfügt über eine weitere Instanzvariable `children`, welche ein Behälter für die Kinderknoten ist.

Um den Ausdruck auszuwerten, wird `event isValid: samState` aufgerufen. Der ODER-Operator in dem Beispiel ruft daraufhin für seine Kindknoten `isValid: samState` auf (mit dem `samState`, den er übergeben bekommen hat) und verknüpft die booleschen Ergebnisse mit dem logischen Oder um selbst `true` oder `false` zurückgeben zu können. Analoges geschieht zuvor für den UND-Operator. Konkret sieht die Implementierung wie folgt aus:

Listing 6.1: AAFBoolOperator isValid:

```
AAFBoolOperator isValid: aSamState

  (type == #operatorAnd) ifTrue: [
      self assert: children size = 2.
      ^ (children first isValid: aSamState) and:
        [children second isValid: aSamState]
  ].

  (type == #operatorOr) ifTrue: [
      self assert: children size = 2.
```

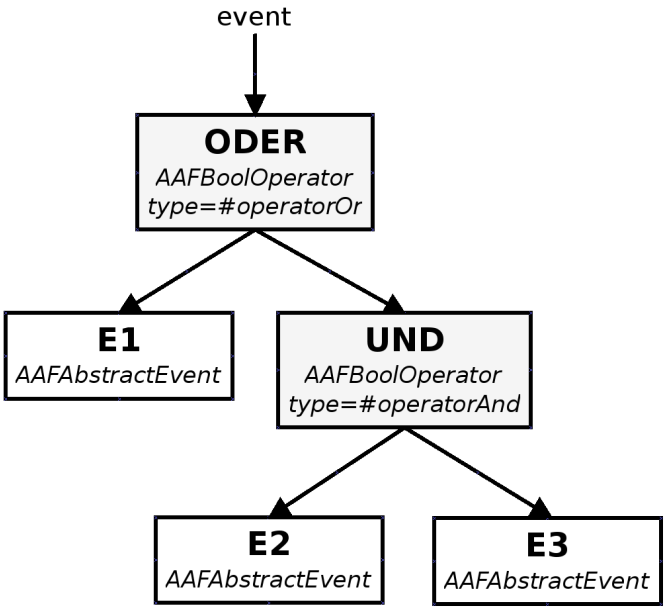


Abbildung 6.1: Repräsentation eines UND/ODER-Ausdrucks zur Laufzeit. Die Rechtecke sind Objekte, die Pfeile Referenzen.

```
    ^ (children first isValid: aSamState) or:
      [children second isValid: aSamState]
].
```

Durch die Verwendung der Methoden `and:` und `or:` bei der Auswertung ist sichergestellt, dass nicht mehr als notwendig ausgewertet wird[1].

6.1.2 *Repräsentation in der Konfigurationsdatei*

Der UND/ODER-Ausdruck muss ebenfalls in der Konfigurationsdatei eines Agenten bzw. einer Funktion abgespeichert werden. Weil die XML-Struktur einer Baumstruktur entspricht, kann der UND/ODER-Ausdruck relativ einfach in XML abgebildet werden.

6.1.2.1 *Format der Konfigurationsdatei*

Das bisherige Format der Konfigurationsdatei wird in Fuhrmann (2010), Abschnitt 4.6 beschrieben. Es wurden folgende Ergänzungen vorgenommen.

Unter dem Tag `agent` wird das neue Tag `activation` eingeführt. Unterhalb dessen wird mit den weiteren Tags `event` und `eventoperator` der Ausdruck spezifiziert. Das Attribut `type` gibt beim Tag `event` den Klassennamen des Ereignisses und beim

1 Wird z. B. der Empfänger von `and:` zu `false` ausgewertet, muss das Argument nicht ausgewertet werden, da bereits feststeht, dass der Ausdruck nicht mehr zu `true` evaluieren kann (bekannt als *Short-circuit evaluation*).

Tag `eventoperator` den Verknüpfungsoperator an. Unterhalb von `event` werden mit `eventprop` die Parameter der Ereignisse abgespeichert. Der Name des Parameters entspricht dem Namen der Instanzvariablen innerhalb der Klasse. Um den Vorgang beim Wiederherstellen des Ereignis-Objekts aus der Konfigurationsdatei zu vereinfachen, wird für jeden Parameter auch dessen Typ abgespeichert.

Das folgende Listing zeigt den bereits bekannten Ausdruck $E_1 \vee (E_2 \wedge E_3)$ mit konkreten Ereignissen.

Listing 6.2: UND/ODER-Ausdruck im XML-Format

```
...
<agent type="AAFInputDistributionAgent">
  ...
  <activation>
    <eventoperator type="and">
      <event type="AAFDrivingOnTrack">
        <eventprop name="minSensorsOnTrack" type="integer"
            value="8"/>
      </event>
      <eventoperator type="or">
        <event type="AAFDrivingTooFast">
          <eventprop name="toleranceBuffer" type="integer"
              value="2"/>
        </event>
        <event type="AAFDrivingTooSlow">
          <eventprop name="toleranceBuffer" type="integer"
              value="2"/>
        </event>
      </eventoperator>
    </eventoperator>
  </activation>
  ...
</agent>
...
```

6.1.2.2 *Schreiben der Konfigurationsdatei*

Mit der Methode `AAFAgent prinXMLOn: writer` wird die Konfigurationsdatei geschrieben. Entsprechend wurde in dieser Methode auch der Quelltext eingeführt, um den UND/ODER-Ausdruck zu speichern:

Listing 6.3: Schreiben der XML-Repräsentation eines Ausdrucks

```
AAFAgent printXMLOn: writer

  ...
  "agent activation"
  writer startTag: 'activation'; endTag.
  writer stream cr.
```

```
self class printXMLEventNodeRec: writer node: self event.
writer endTag: 'activation'.
writer stream cr.
...
```

Das eigentliche Schreiben übernimmt die rekursive Klassenmethode `printXMLEventNodeRec: writer node: self event`. Die Rekursion findet über die einzelnen Knoten des Baums statt (siehe Abschnitt 6.1.1 weiter oben).

6.1.2.3 *Lesen der Konfigurationsdatei*

Das Lesen der Konfigurationsdatei wird in `AAFXMLParser` vorgenommen. Um den gespeicherten UND/ODER-Ausdruck in eine Laufzeit-Repräsentation (siehe Abschnitt 6.1.1 weiter oben) zu überführen, wurden die Methoden `startElement:attributeList:` und `endElement:` entsprechend erweitert[2].

6.1.3 *Repräsentation in dem Automatiken-GUI*

Das Automatiken-GUI wurde dahingehend erweitert, dass der Benutzer einen UND/ODER-Ausdruck angezeigt bekommt und diesen bearbeiten kann. Im Konfigurationsbereich des Automatiken-GUI befindet sich an oberster Stelle das Fenster „Aktivierung“. Abbildung 6.2 zeigt die GUI-Repräsentation für den in Abschnitt 6.1.2 vorgestellten UND/ODER-Ausdruck aus der Konfigurationsdatei.

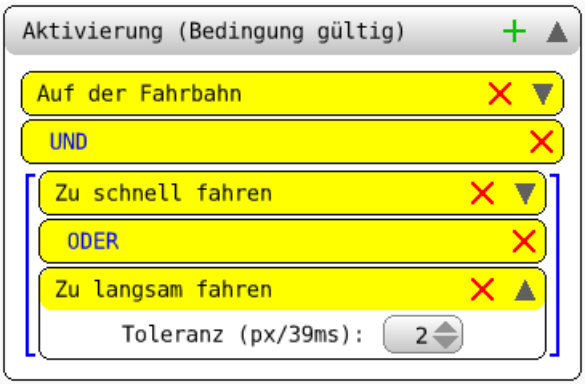


Abbildung 6.2: Die Ereignis-Konfiguration befindet sich in dem Automatiken-GUI im Konfigurationsbereich einer Funktion.

Die Benutzung dieser Ereignis-Konfiguration ist in der Bedienungsanleitung im Abschnitt C.4.7 dokumentiert.

Wie in Abschnitt 2.3.1 beschrieben, werden Morphe in einer Baumstruktur organisiert. Daher kann die Laufzeit-Repräsentation

2 Die Änderungen sind zu umfangreich, als dass sie hier in Kürze wiedergegeben werden können.

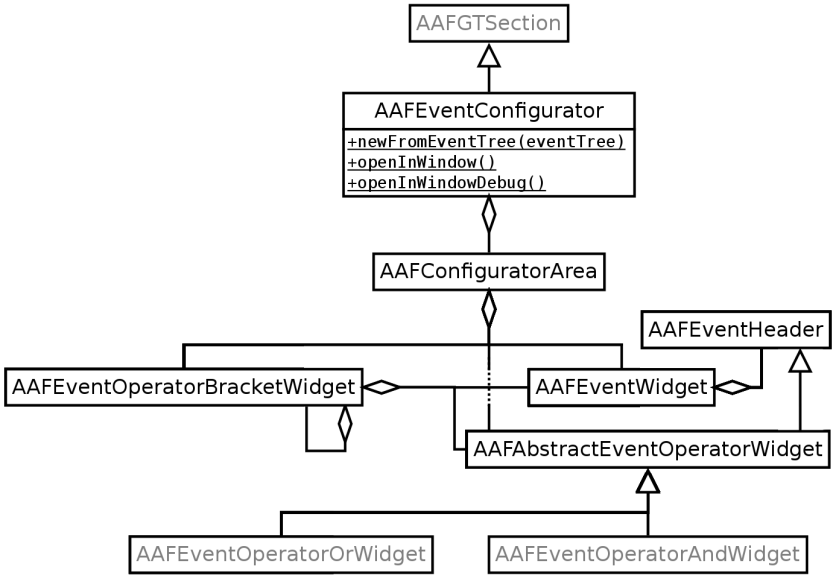


Abbildung 6.3: Klassendiagramm der Ereignis-Konfiguration (Kategorie „AAFGT-Events-Configurator“).

(siehe Abschnitt 6.1.1 weiter oben) relativ direkt in dem GUI widergespiegelt werden.

Die Ereignis-Konfiguration ist in der Klassenkategorie `AAFGT-Events-Configurator` umgesetzt. Die dort enthaltenen Klassen stehen wie in Abbildung 6.3 gezeigt miteinander in Beziehung. Auf die Klassen in schwarzer Schrift wird im Folgenden näher eingegangen.

- `AAFEventConfigurator` (Haupt-Widget)
  Das ist die Hauptklasse der Ereignis-Konfiguration. Mit der Klassenmethode `newFromEventTree: eventTree` wird ein neues Objekt erzeugt, welches die übergebene Laufzeit-Repräsentation des UND/ODER-Ausdrucks in ein GUI überführt. Die beiden Klassenmethoden `openInWindow` und `openInWindowDebug` ermöglichen es, das GUI in einem Squeak-Fenster zu öffnen, ohne eine Laufzeit-Repräsentation zu übergeben. Letztere Methode dient dem manuellen Testen und fügt automatisch alle bekannten Ereignisse hinzu.
  Der `AAFEventConfigurator` ist ein `AAFGTSection` und ist aus einem *Header* und einer *Area* aufgebaut (analog zu einem `AAFEventWidget`). Der Header kann auf der rechten Seite Buttons enthalten. Der `AAFEventConfigurator` fügt einen Button (grünes Plus-Symbol) hinzu. Ein Links-Klick darauf öffnet dem Benutzer ein Menü zum Hinzufügen von Operatoren und Ereignissen.

- `AAFConfiguratorArea` (Bereich für Drag and Drop)
  Der `AAFEventConfigurator` verwirft die Standard-*Area* und ersetzt diese durch eine `AAFConfiguratorArea`, um Drag and

Drop zu ermöglichen. Eine `AAFConfiguratorArea` kann alle drei möglichen Typen von Widgets enthalten, die der Benutzer auch zu Gesicht bekommt: Ereignis-Widgets, Operator-Widgets und Klammer-Widgets.

- `AAFEventOperatorBracketWidget` (Klammer-Widget)
  Die blauen Klammern des Klammer-Widgets sind in Abbildung 6.2 zu sehen. Das Klammer-Widget kann Ereignis-Widgets, Operator-Widgets oder Objekte von selbigen Typ enthalten. Letzteres ermöglicht theoretisch[3] die Darstellung beliebiger Schachtelungstiefen.

- `AAFEventHeader` (Header-Widget)
  Der Event-Header stellt die gelben abgerundeten Rechtecke dar, mit denen der Benutzer interagieren kann. Auf der linken Seite wird ein Text angezeigt, auf der rechten Seite verschiedene Buttons. Der Standard-Button zum Entfernen des Widgets, dargestellt durch ein rotes 'X', ist per Voreinstellung vorhanden.

- `AAFAbstractEventOperatorWidget` (Operator-Widget)
  Das Operator-Widget verändert die Voreinstellungen des `AAFEventHeader` dahingehend, dass es die Operatoren in blauer Farbe eingerückt darstellt, damit sich diese von den Ereignis-Widgets unterscheiden. Konkrete Klassen für das Operator-Widget sind `AAFEventOperatorOrWidget` und `AAFEventOperatorAndWidget`.

- `AAFEventWidget` (Ereignis-Widget)
  Das Ereignis-Widget besteht aus einem `AAFEventHeader` und einer *Area*. Falls die *Area* von `AAFAbstractEventGUI` zur Verfügung gestellt wird, dann erscheint im `AAFEventHeader` ein weiterer Button (ausgefülltes dunkles Dreieck), der die *Area* (und damit die Parameter der Ereignisse) auf- und bei erneutem Links-Klick wieder zuklappen kann.

Das Objekt vom Typ `AAFEventConfigurator` wird in der Klassenmethode `AAFAgentDialog newForNode:withMainPanel:` erzeugt:

Listing 6.4: Erzeugen von AAFEventConfigurator

```
AAFAgentDialog newForNode: anAAFNode withMainPanel: mainPanel
  "Sets up dialog for a certain node."

  ...
  instance := self basicNew initialize.
  anAAFAgentOrGraph := anAAFNode delegate.
```

3 Praktisch ist die Schachtelungstiefe durch die Breite des Owner-Morphen limitiert. Es ist jedoch davon auszugehen, dass nicht mehr als einmal geschachtelt wird.

```
  instance delegate: anAAFAgentOrGraph.

  instance
    eventConfigurator:
      (AAFEventConfigurator newFromEventTree:
          anAAFAgentOrGraph event);
  ...
```

Die Methode wird vom Automatiken-GUI aufgerufen, sobald der Benutzer eine Funktion im Ablaufplan anklickt. Mit der Klassenmethode `newFromEventTree:` wird dem `AAFEventConfigurator` die Laufzeit-Repräsentation übergeben. Aus den Ereignis-Objekten werden die einzelnen Widgets generiert und dargestellt. Ob Ereignisse über Konfigurationsmöglichkeiten verfügen, wird herausgefunden, indem die entsprechende GUI-Klasse (Name der Ereignisklasse + 'GUI') konsultiert wird.

6.1.4 *Einbindung in die Funktionen*

Der UND/ODER-Ausdruck für eine Funktion soll bestimmen, ob die Funktion ausgeführt wird oder nicht. Es stellt sich die Frage, an welcher Stelle der UND/ODER-Ausdruck ausgewertet wird. Zwei Ansätze sind denkbar:

1. Der UND/ODER-Ausdruck wird *außerhalb* der Methode `compute:` ausgewertet und das Ergebnis entscheidet über die Ausführung von `compute:`.
2. Der UND/ODER-Ausdruck wird *innerhalb* der Methode `compute:` ausgewertet.

Der erste Ansatz scheint auf den ersten Blick die bessere Lösung zu sein, da die `compute:`-Methoden der einzelnen Funktionen nicht angepasst werden müssen. Der zweite Ansatz wurde aber aus den folgenden Gründen bevorzugt. Einerseits kann eine Funktion Log-Ausgaben generieren, obwohl sie nicht ausgeführt werden soll. Andererseits sind Funktionen denkbar, die davon ausgehen (müssen), dass sie in jedem Simulationsschritt ausgeführt werden.

Daher sollte die Methode `compute:` jeder Funktion wie folgt aussehen:

Listing 6.5: Abfrage des Ereignis-Systems

```
MyAAFAgent compute: aSamState

  (self event isValid: aSamState)
    ifFalse: [ ^ aSamState ].
  ...
  ^ aSamState
```

Mit `self event isValid:` wird der UND/ODER-Ausdruck für den aktuellen SamState ausgewertet. Wenn der Ausdruck zu `false` evaluiert, soll die eigentliche Funktionalität (symbolisiert durch die drei Punkte im Listing) nicht ausgeführt werden. Das wird erreicht, in dem der übergebene SamState wieder zurückgegeben wird. Vor Rückgabe des SamState können, je nach Funktionen, Log-Ausgaben oder andere Operationen ausgeführt werden, die nicht der „Kern-Aktivität" der Funktion entsprechen.

6.1.5 *Limitierung der oberen Schranke*

Dem Ereignis-Modell (vgl. 5.3) nach kann die obere Schranke bei Ereignissen mit Streckenbezug beliebig weit nach der obersten Pixelzeile des Streckenelements liegen. Die Umsetzung entspricht in diesem Aspekt nicht vollständig dem Modell, da die oberste Pixelzeile des Streckenelements die obere Schranke limitiert (vgl. Abschnitt C.13.1). Diese Einschränkung hat die Implementierung vereinfacht, weil dadurch die Positionen der bereits „vergangenen" Streckenelemente (d. h. hinter dem Objekt liegende Streckenelemente) nicht gemerkt werden müssen, nachdem das Objekt diese durchfahren hat.

6.2 EREIGNISSE

Während alle Ereignisse mit Streckenbezug implementiert werden konnten, konnten in der gegebenen Zeit nur einige Ereignisse mit Bezug zum Fahrverhalten implementiert werden (vgl. 5.4.2):

- Nicht implementiert sind die Ereignisse „Abstand zum Fahrbahnrand", „Differenz der Joystick-Eingaben", „Lenkung in gewünschte Richtung" und „Ungenaues Fahren".
- Das Ereignis „Geschwindigkeitsempfehlung" wurde in vier einzelnen Ereignissen implementiert: „Zu schnell fahren", „Zu langsam fahren", „Bremsempfehlung (dyn. Hindernis)" und „Beschleunigungsempfehlung (dyn. Hindernis)". Die einzelnen Ereignisse haben den Vorteil, dass ihr Name in der Ereignis-Konfiguration aussagekräftiger ist[4]. Außerdem sind die Zuständigkeiten der einzelnen Ereignisse klarer.
- Das Ereignis „Kollisionskurs mit Hindernissen" wurde nur zum Teil implementiert. Eine Kollisionserkennung ist in der Klasse `AAFSpeedHintsAtDynamicObstacles` implementiert. Es werden nur Kollisionen mit dynamischen Hindernissen erkannt und die Ereignisse „Bremsempfehlung (dyn.

4 Bei „Geschwindigkeitsempfehlung" müsste der Benutzer erst die Parameter einsehen, um zu verstehen, wann genau das Ereignis eintritt.

Hindernis)“ und „Beschleunigungsempfehlung (dyn. Hindernis)“ benutzt diese. Die Erkennung von Kollisionen mit statischen Hindernissen ist nicht implementiert.

Für Benutzer sind die einzelnen Ereignisse in der Bedienungsanleitung des Automatiken-GUI in Abschnitt C.13 dokumentiert. Für Implementierer neuer Ereignisse dokumentiert der Anhang B die Entwicklung und Integration neuer Ereignisse.

Jedes Ereignis wurde durch eine Klasse in der Kategorie „AAF-Events“ implementiert. Für jedes Ereignis ist in Tabelle 6.1 die entsprechende Ereignisklasse zugeordnet. Ereignisklassen mit Streckenbezug haben den Präfix „`AAFTrack`“ im Namen. Ereignisklassen mit Bezug zum Fahrverhalten haben den Präfix „`AAFDriving`“ im Namen. Die Klasse `AAFAlwaysTrueEvent` stellt einen Sonderfall dar und steht für das künstliche Ereignis „Immer aktiv“[5].

EREIGNIS	KLASSE
Immer aktiv	`AAFAlwaysTrueEvent`
Bevor/im Gabelungsbereich	`AAFTrackForkAhead`
Bevor/im Hindernisbereich	`AAFTrackObstacleAhead`
Bevor/im Kurvenbereich	`AAFTrackCurveAhead`
Bremsempfehlung (dyn. Hindernis)	`AAFDrivingDynObsBrakeNow`
Beschleunigungsempfehlung (dyn. Hindernis)	`AAFDrivingDynObsSpeedUpNow`
Zu schnell fahren	`AAFDrivingTooFast`
Zu langsam fahren	`AAFDrivingTooSlow`
Auf der Fahrbahn	`AAFDrivingOnTrack`
Eingabe-Timeout	`AAFDrivingWithoutInput`
Auf einer Seite der Mittellinie	`AAFDrivingOneSideOfRacingLine`

Tabelle 6.1: Ereignisse und die entsprechenden Ereignisklassen in der Klassenkategorie „AAF-Events“.

Es gibt zwei Klassen, von denen die Ereignisse abgeleitet sein können:

1. Die Klasse `AAFAbstractEvent` ist die abstrakte Basisklasse aller Ereignisklassen. Für die Implementierung von Ereignisklassen mit Bezug zum Fahrverhalten, wird von dieser Klasse abgeleitet[6].

5 Dieses Ereignis ist die Voreinstellung in der Ereignis-Konfiguration (siehe Abschnitt 6.1.3 weiter oben).

6 Die Klasse `AAFAlwaysTrueEvent` ist ebenfalls von `AAFAbstractAgent` abgeleitet, hat aber kein Bezug zum Fahrverhalten, da es sich um ein künstliches Ereignis handelt.

2. Die Klasse `AAFAbstractTrackAreaEvent` ist von der Klasse `AAFAbstractEvent` abgeleitet und ebenfalls abstrakt. Für die Implementierung von Ereignisklassen mit Streckenbezug, wird von dieser Klasse abgeleitet.

Abbildung 6.4 zeigt das Klassendiagramm der Ereignisklassen.

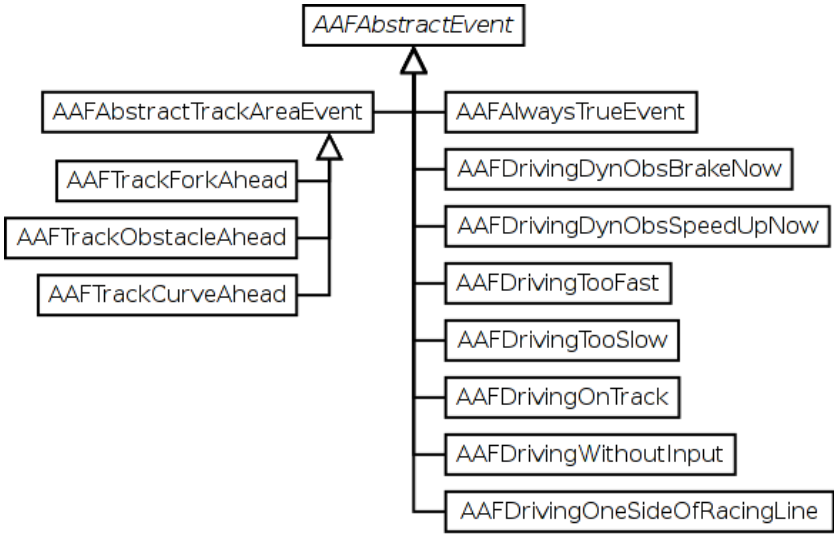


Abbildung 6.4: Klassendiagramm für die Ereignisklassen.

Zu einer Ereignisklasse `E` gibt es die Klassen `ETest` und `EGUI`. `ETest` ist der gleichen Klassenkategorie wie `E` zugeordnet und implementiert Tests für `E` (siehe Abschnitt 7.2). `EGUI` ist der Klassenkategorie „AAFGT-Events“ zugeordnet und implementiert das GUI für `E`. Die GUI-Klassen werden in der Ereignis-Konfiguration (siehe Abschnitt 6.1.3 weiter oben) benutzt.

In den folgenden Unterabschnitten werden Details zur praktischen Umsetzung der einzelnen Ereignisse dokumentiert.

6.2.1 *Implementierung der Ereignisse mit Streckenbezug*

Die wesentliche Information für Ereignisse mit Streckenbezug ist die Distanz in Pixel von dem Objekt zu den Streckenelementen (Gabelung, Hindernis und Kurve). Die Funktionalität zur Bestimmung dieser Distanzen ist in der Klasse `SAMControllerDistance` (Klassenkategorie „ATEO-SAM-Controller“) implementiert. Da die ermittelten Distanzen relativ häufig abgerufen werden können und die Berechnungen nicht mehrfach durchgeführt werden sollen, sind sie in dem assoziativen Array `distanceDictionary` eines SamStates für jeden Simulationsschritt abgelegt. Die Methode `updateDistances` wird dazu in `SAMControllerStep processStep` aufgerufen. Die Schlüssel von `distanceDictionary` sind Symbole, die einen bestimmten Gabelungs-, Hindernis- oder Kurventyp angeben. Beispielsweise liefert der Schlüssel `#nextDynamicObstacle`

den Abstand zum nächsten dynamischen Hindernis in Pixel. Der Abstand wird ausgehend von der obersten Pixelzeile des Objekts bis zur untersten Pixelzeile des Streckenelements bestimmt. Die einzelnen Ereignisse mit Streckenbezug sind so implementiert, dass sie für die vom Benutzer bestimmten Gabelungs-, Hindernis- oder Kurventypen auf die entsprechenden Schlüssel des `distanceDictionary` abbilden und somit an die richtige Distanz kommen.

In den Abschnitten für die Ereignisse mit Streckenbezug wird nachfolgend beschrieben, wie die Distanzen berechnet werden. Dabei handelt es sich genau genommen um die Beschreibung Methoden aus `SAMControllerDistance`, da in den Ereignisklassen die Berechnung nicht stattfindet.

6.2.1.1 *Ereignis „Bevor/im Gabelungsbereich“*

Die Positionen der Gabelungskacheln für die aktuelle Strecke sind bereits in den beiden Listen `elrForks` und `rllForks` des SAMModelData-Objekt gespeichert. In `SAMControllerDistance distanceToNextFork:` wird die Distanzberechnung durchgeführt. Mit Hilfe der aktuellen Position des Objekts auf der Strecke wird das richtige Listenelement (d. h. die nächste Gabelung des entsprechenden Typs) iterativ gefunden, sodass die Distanz bestimmt werden kann.

6.2.1.2 *Ereignis „Bevor/im Hindernisbereich“*

Die Positionen der Hindernisse für die aktuelle Strecke sind bereits in der Liste `obstaclesLeft` des SAMModelData-Objekts gespeichert. In `SAMControllerDistance distanceToNextObstacle:` wird die Distanzberechnung durchgeführt. Mit Hilfe der aktuellen Position des Objekts auf der Strecke wird das richtige Listenelement (d. h. der richtige Typ) iterativ gefunden, sodass die Distanz bestimmt werden kann.

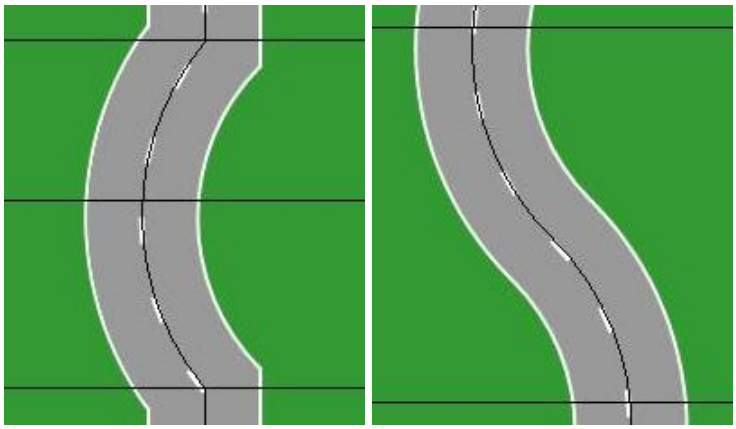
Die beiden Hinderstypen „Statischer 25%-Slalom“ und „Statischer 50%-Slalom“ sind nicht in den Anforderungen (vgl. Abschnitt 5.4.1) zu finden, da diese nachträglich für die Hinweisfunktion „Hindernis-Slalom“ (vgl. Abschnitt 6.3.3) eingeführt wurden.

6.2.1.3 *Ereignis „Bevor/im Kurvenbereich“*

Die Positionen von Kurven sind nicht im SAMModelData-Objekt zur Verfügung gestellt, da diese bisher nicht benötigt wurden.

Es müssen Kurven verschiedenster Krümmungen erkannt werden. Team 37 möchte „Links- und Rechtskurven und verschiedene Krümmungen (visuell)“ ankündigen und Team 46 sieht eine „Warnung vor scharfe[n] Kurve[n]“ vor.

Dreht der Fahrer eines PKWs das Lenkrad nach links um das Fahrzeug weiter auf der Fahrspur zu halten, fährt er offensichtlich durch eine Linkskurve. Lenkt er nach rechts, fährt er durch eine Rechtskurve. Dieses Lenkverhalten ist für die beiden Streckenabschnitte in Abbildung 6.5 angemessen. Die MWB in SAM nehmen die Strecke allerdings nicht aus dem Inneren des Objekts wahr (wie im gewöhnlichen Fahrzeug). Sie sehen das Objekt sowie die Fahrbahn von oben. Während aus ihrer Perspektive der Streckenabschnitt in Abbildung 6.5a noch eine Links-Rechts-Kurve darstellt, handelt es sich bei dem Streckenabschnitt in Abbildung 6.5b für sie um eine Linkskurve - zu keinem Zeitpunkt ist es für sie erforderlich, nach rechts zu lenken. Eine Warnung vor einer Rechtskurve wäre bei diesem Streckenabschnitt daher irreführend. Die übliche Definition einer Links- und Rechtskurve wird beibehalten, allerdings muss die Perspektive der MWB miteinbezogen werden.



(a) (b)

Abbildung 6.5: Auf Streckenabschnitt (b) müssen die MWB nur nach links lenken, also handelt es sich um eine Linkskurve.

Der Krümmungen der verschiedenen Kurven werden durch ihre Breite unterschieden. Bei Kurven mit geringer Breite muss weniger lange in eine Richtung gelenkt werden, als bei Kurven großer Breite. Die Kurvenerkennung ist mit Hilfe der Mittellinie und einer Zustandsmaschine implementiert. Ist für eine beliebige Strecke die Mittellinie bekannt, so können die Kurven auf dieser erkannt werden. Die Mittellinie wird vollständig durchlaufen und dabei werden verschiedene Zustände angenommen, die auch den verschiedenen Kurvensektionen entsprechen. Verlaufen die Koordinaten immer weiter nach links oder gerade aus, dann handelt es sich um eine Linkskurve. Verlaufen sie nach rechts, dann handelt es sich um eine Rechtskurve. Gerade Fahrbahnabschnitte haben mehr als 20 aufeinanderfolgende Koordinaten mit gleichen X-Wert - sobald das erkannt ist, muss es

sich um eine geraden Abschnitt handeln. Die Implementierung ist in Klasse `AAFTrackCurves` zu finden. Mit der Klassenmethode `drawRacingLineAndCurveSectionsForAllTracks` werden auf jede Strecke die Mittellinie sowie die Grenzen der verschiedenen Kurvensektionen gezeichnet. Die Bilder werden im gleichen Verzeichnis wie die Squeak.exe abgelegt. Abbildung 6.6 zeigt einen Ausschnitt aus einer solchen generierten Bild-Datei.

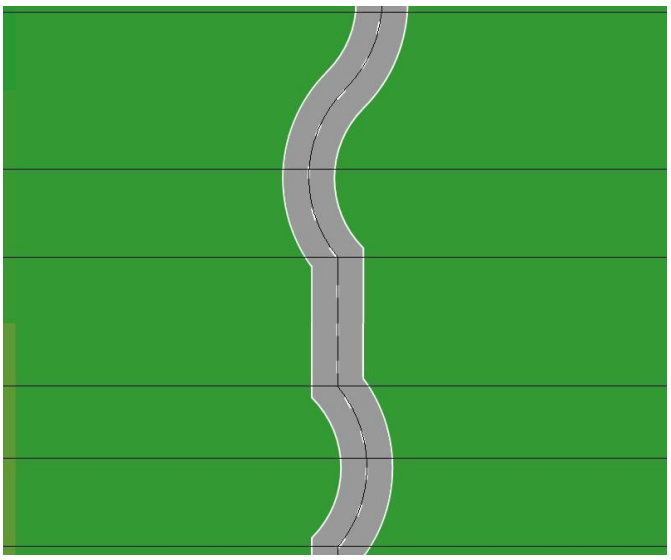


Abbildung 6.6: Erkannte Sektionen sind durch horizontale schwarze Linien getrennt. Die Mittellinie ist erkennbar.

Es konnten insgesamt vier verschiedene Kurvenbreiten identifiziert werden. Abbildung 6.7 zeigt alle entdeckten Kurvenbreiten für die Linkskurve (für die Rechtskurve wurden die gleichen gefunden). Dem Benutzer werden diese verschiedenen Kurvenbreiten als „kurz“, „mittel“, „lang“ und „sehr lang“ präsentiert (siehe Abschnitt C.13.4), anstatt die Breiten direkt anzugeben.

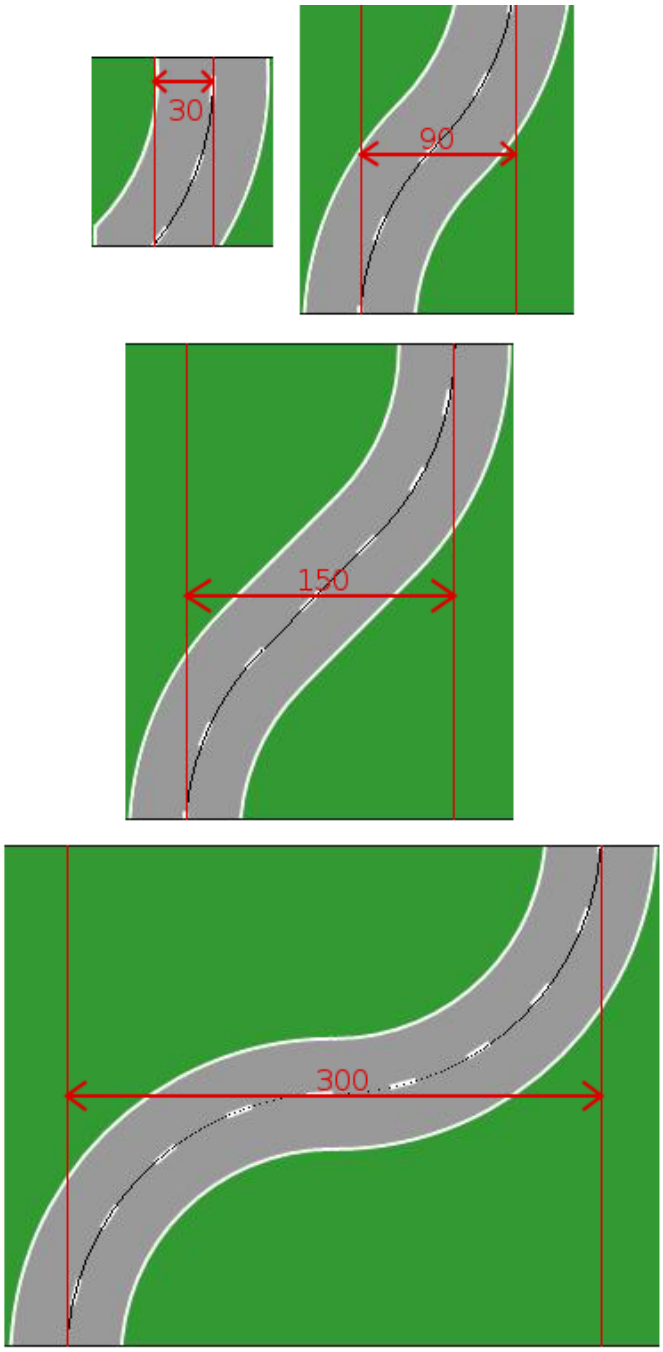


Abbildung 6.7: Die verschiedenen Kurvenbreiten in Pixel für die Rechtskurve (analog für die Linkskurve).

In `SAMControllerDistance` wird die Klassenmethode `AAFTrackCurves curveSectionsForRacingLine:` benutzt, um an eine Liste der Kurvensektionen für die aktuelle Mittellinie zu gelangen. Schließlich können damit auch die Distanzen für verschiedene Kurven berechnet und in dem `distanceDictionary` abgelegt werden.

6.2.2 *Ereignis „Bremsempfehlung (dyn. Hindernis)“*

Die eigentliche Funktionalität für die Entscheidung, ob eine Bremsempfehlung signalisiert werden soll oder nicht, ist in der Klasse `AAFSpeedHintsAtDynamicObstacles` zu finden.

Mit der Klassenmethode `collisionWithCurrentSpeedForState:` wird abgefragt, ob bei gleichbleibender Richtung und Geschwindigkeit eine Kollision stattfinden wird. Wenn die Kollision festgestellt wird, dann tritt das Ereignis ein, da die Kollision immer durch Stillstand verhindert werden kann. Ausgenommen davon die Fälle, in denen das Objekt bereits auf Höhe des dynamischen Hindernisses ist.

Für die Kollisionserkennung werden das Objekt und das dynamische Hindernis mit ihren aktuellen Richtungen und Geschwindigkeiten iterativ weiterbewegt, bis sie aufeinanderstoßen oder sich verfehlen.

Es wird angenommen, dass die MWB ihre Joystick-Lenkungen von einem Simulationsschritt zum nächsten anpassen können (von der aktuellen Geschwindigkeit auf Stillstand).

6.2.3 *Ereignis „Beschleunigungsempfehlung (dyn. Hindernis)“*

Dieses Ereignis ist ähnlich zu dem Ereignis „Bremsempfehlung (dyn. Hindernis)“ (siehe Abschnitt 6.2.2 oben) implementiert. In der Klasse `AAFSpeedHintsAtDynamicObstacles` ist die eigentliche Funktionalität zu finden.

Mit der Klassenmethode `collisionWithMaxSpeedForState:` wird abgefragt, ob bei gleichbleibender Richtung und Maximalgeschwindigkeit (nicht aktuelle Geschwindigkeit) eine Kollision stattfinden wird. Die Kollisionserkennung erfolgt dabei wie bei Abschnitt 6.2.2 erklärt. Wenn keine Kollision festgestellt wird, dann tritt das Ereignis ein.

Auch bei diesem Ereignis wird angenommen, dass die MWB ihre Joystick-Lenkungen augenblicklich anpassen können.

6.2.4 *Ereignis „Zu schnell fahren“*

Die aktuelle Geschwindigkeit des Objekts wird durch `AAFValues getTrackingObjectCurrentSpeed:` ermittelt. Die ideale Geschwindigkeit eines Objekts, welches sich auf der Mittellinie befindet, wird durch die Methode `calculateSpeedForY:branch:` der Klasse `AAFTrackOptimalSpeed` bereitgestellt.

Durch Vergleich der beiden Geschwindigkeiten sowie der vom Benutzer eingestellten Toleranz, kann bestimmt werden, ob das Ereignis eintritt oder nicht.

Die beiden genannten Klassen(methoden) wurden von Helmut Weidner-Kim implementiert und sind Teil seiner Diplomarbeit Weidner-Kim (2012).

6.2.5 *Ereignis „Zu langsam fahren“*

Analog zum Ereignis „Zu schnell fahren“, siehe Abschnitt 6.2.4.

6.2.6 *Ereignis „Auf der Fahrbahn“*

Über `noSensorsOffTrack` des SamStates wird abgefragt, wie viele Sensoren für den aktuellen Simulationsschritt signalisieren, dass das Objekt abseits der Fahrbahn ist. Da es nur acht Sensoren gibt, kann bestimmt werden, wie viele Sensoren entsprechend das Gegenteil signalisieren. Ist diese Anzahl größer oder gleich der vom Benutzer eingestellten Anzahl, dann tritt das Ereignis ein.

6.2.7 *Ereignis „Eingabe-Timeout“*

Über den übergebenen SamState werden die Joystick-Eingaben mit `joystickRaw1` und `joystickRaw2` abgefragt. Die Eingabe `(-1@ -1)` entspricht der Joystick-Ruhestellung. Über eine Instanzvariable wird gespeichert, wie viele Simulationsschritte in Folge die Ruhestellung signalisiert wird. Da ein Simulationsschritt 39ms dauert, kann bestimmt werden, ob die eingestellte Zeit verstrichen ist und ob das Ereignis eintritt oder nicht.

6.2.8 *Ereignis „Auf einer Seite der Mittellinie“*

Der X-Wert des Mittelpunkts des Objekts wird mit dem X-Wert der Mittellinie auf gleicher Höhe (Y-Wert) verglichen. Die Mittellinie wird als der gewünschten Seite zugehörig zugeordnet.

Die Mittellinie steht über ein Singleton-Objekt zur Verfügung (`AAFSupportRacingLine getInstance`). Der zu einem Y-Wert gehörende X-Wert kann mit Hilfe der Methode `racingLine: y` bestimmt werden.

Der X-Wert der Position des Objekt (linke obere Ecke) wird durch `AAFValues getTrackingObjectXPosition:` ermittelt. Aus historischen Gründen gibt diese Methode den Wert als negative Zahl zurück. Daher wird der Betrag genommen und mit fünfzehn addiert, um den X-Wert des Mittelpunkts zu bestimmen.

Die genannten Klassen(methoden) wurden von Helmut Weidner-Kim implementiert und sind Teil seiner Diplomarbeit Weidner-Kim (2012).

6.3 HINWEISFUNKTIONEN

Die Benutzerdokumentation für die Hinweisfunktionen ist in der Bedienungsanleitung des Automatiken-GUI in Abschnitt C.12 dokumentiert. Mit den dort beschriebenen Funktionen wurden die Funktionen aus den Konzeptbögen (vgl. Kapitel 4) implementiert. In Anhang A werden die implementierten Funktionen aufgelistet.

Tabelle listet die implementierten Hinweisfunktionen und ihre zugehörigen Klassen auf.

HINWEISFUNKTION	KLASSE
Hinweis (auditiv)	`AAFGenericAuditiveHintAgent`
Hinweis (visuell)	`AAFGenericVisualHintAgent`
Gabelungshinweis (auditiv)	`AAFForkAuditiveHintAgent`
Gabelungshinweis (visuell)	`AAFForkVisualHintAgent`
Hindernisslalom (visuell)	`AAFObstacleVisualHintAgent`
Streckenvorschau (visuell)	`AAFMapPreviewAgent`

Tabelle 6.2: Hinweisfunktionen und die entsprechenden Klassen in der Klassenkategorie „AAF-Agents-Concepts-Visual“.

Jede Klasse ist von `AAFAgent` abgeleitet und wurde der Klassenkategorie „AAF-Agents-Concepts-Visual“ zugeordnet. Klassen, die visuelle Hinweise implementieren, haben „VisualHint“ im Namen. Klassen, die auditive Hinweise implementieren, haben „AuditiveHint“ im Namen.

Zu einer Klasse `F` gibt es die Klasse `FDialog`, in welcher das GUI für die Funktion implementiert ist. Die GUI-Klassen sind in der Kategorie „AAFGT-Dialog“ zu finden.

In den folgenden Unterabschnitten werden Details zur praktischen Umsetzung der einzelnen Funktionen und ihrer Klassen dokumentiert.

6.3.1 *Auditive Hinweisfunktionen*

Die auditiven Hinweisfunktionen benutzen die Squeak-Klasse `StreamingMP3Sound`[7], um eine MP3-Datei abzuspielen. Die Methode `compute:` sieht entsprechend sehr einfach aus:

Listing 6.6: AAFGenericAuditiveHint compute:

```
AAFGenericAuditiveHint compute: aSamState

  (self event isValid: aSamState)
    ifFalse: [ ^ aSamState ].
```

7 vgl. 2.7 im Kapitel „Voraussetzungen“

```
sound isPlaying
  ifFalse: [ sound play ].

^ aSamState
```

Der übergebene SamState wird nicht verändert. Da in SAM keine Sound-Objekte verwaltet werden, wird die Datei direkt aus dieser Funktion abgespielt.

Die Funktion „Gabelungshinweis (auditiv)“ benutzt intern zwei „Hinweis (auditiv)“-Funktionen, weil ein auditiver Hinweis für den linken und rechten Zweig benötigt wird.

Alle in Abschnitt 4.3.1 aufgelisteten abstrakten Töne und Texte wurden generiert, im Web unter einer passenden Lizenz gefunden, oder vom Verfasser gesprochen[8].

Die dem Verzeichnis „resource.sounds/Quellen.txt“ hinzugefügten Sound-Dateien für die abstrakten Töne sind „1kh ton.mp3“, „bing.mp3“, „drei mal tuten.mp3“, „hupen.mp3“, „piepton 125ms.mp3“, „piepton 250ms.mp3“, „piepton 63ms.mp3“ und „warnton.mp3“. Der „Piepton“, der seine Frequenz abhängig von einer Distanz zu einem Streckenelement ändert, wurde mit den drei Sound-Dateien „piepton 250ms.mp3“, „piepton 125ms.mp3“ und „piepton 63ms.mp3“ realisiert. Die erste Hälfte dieser Sounds spielt einen Ton jeweils der Länge 250ms, 125ms und 63ms ab. Die zweite Hälfte ist jeweils genau so lang und enthält „Stille“. Wird eine der Sound-Dateien mehrmals hintereinander abgespielt, entsteht der Effekt eines Pieptons. Mit drei Funktionen „Hinweis (auditiv)“, die jeweils drei Ereignisse mit Bezug zum selben Streckenelement, aber unterschiedlichen Abständen zu den Streckenelementen eingestellt haben, kann der „Piepton“, der seine Frequenz abhängig von einer Distanz zu einem Streckenelement ändert, realisiert werden.

Für die gesprochenen Texte wurden folgende Sound-Dateien hinzugefügt: „achtung gabelung.mp3“, „achtung hindernis.mp3“, „achtung kurve.mp3“, „achtung scharfe kurve.mp3“, „bremsen.mp3“, „links.mp3“, „nach links.mp3“, „nach rechts.mp3“, „rechts.mp3“, „zu langsam.mp3“ und „zu schnell.mp3“, „erinnerung moeglichst genau.mp3“, „erinnerung moeglichst schnell.mp3“, „folgen sie dem pfeil.mp3“, „langsamer.mp3“, „schneller.mp3“, „vorsicht gabelung.mp3“, „vorsicht hindernis.mp3“.

6.3.2 *Visuelle Hinweisfunktionen*

Die visuellen Hinweisfunktionen erzeugen `BitBlt`-Objekte, die an den übergebenem SamState in `compute:` weitergereicht werden:

8 In der Datei resource.sounds/Quellen.txt wurde der Urpsrung und die Lizenz der neu hinzugefügten Sound-Dateien festgehalten.

Listing 6.7: AAFGenericVisualHint compute:

```
AAFGenericVisualHint compute: aSamState
  ...
  aSamState bitBlts add: bitBlt.
  ...

  ^ aSamState
```

Die Liste `bitBlts` wurde in `SAMModlData` eingeführt, um die `BitBlt`-Objekte der einzelnen Hinweisfunktionen zu sammeln. Die Zeichenoperationen, die in den `BitBlt`-Objekten gekapselt sind, könnten zwar direkt in der Methode `compute:` ausgeführt werden, aber die Resultate wären nicht sichtbar, da SAM mit seinen eigenen `BitBlt`-Objekten (Strecke, Objekt, Hindernisse) die Zeichnungen der Funktionen überdecken würde[9].

Die Funktion „Gabelungshinweis (visuell)“ benutzt intern vier „Hinweis (visuell)“-Funktionen, da jeweils zwei Bilder für die zwei verschiedenen Gabelungen benötigt werden (potenziell).

6.3.3 *Hindernisslalom*

Diese Funktion ist bei den Anforderungen (vgl. Abschnitt 5.5) nicht aufgeführt, da der Bedarf an dieser Hinweisfunktion erst nach der Definition der Anforderungen offensichtlich wurde.

Die Umsetzung eines Hinweises bei einem Hindernisslalom hätte auch mit „Hinweis (visuell)“ umgesetzt werde können. Diese Umsetzung hätte aber zwei Probleme. Erstens würde der Hinweis ausgeblendet werden, sobald die oberste Pixelzeile des Objekts weiter als die oberste Pixelzeile des Hindernisses rückt (siehe Abschnitt 6.1.5). Zweitens hätte der Benutzer auf einen Vorschau-Bereich verzichten müssen[10]. Dieser ist bei der Positionierung des Bildes sehr nützlich.

Das erste Problem wurde gelöst, in dem die zwei Hindernistypen „Statischer 25%-Slalom“ und „Statischer 50%-Slalom“ eingeführt wurden. Ein Slalom erstreckt sich von der untersten Pixelzeile des ersten (unteren) Hindernisses bis zur obersten Pixelzeile des zweiten (oberen) Hindernisses.

6.3.4 *Gabelungszweige*

In diesem Abschnitt wird auf die zu empfehlenden Zweige (siehe auch Abschnitt 4.4 auf Seite 36) der beiden Hinweisfunktionen

9 Das liegt an der festgelegten Reihenfolge in `SAMControllerStep processStep`. Es werden zuerst alle Funktionen bzw. Agenten ausgeführt und danach zeichnet SAM die angezeigten Grafiken (Strecke, Objekt, Hindernisse) neu.

10 Die Vorschau-Bereiche sind jeweils speziell für die einzelnen visuellen Hinweisfunktionen implementiert.

für Gabelungen eingegangen. Die gewünschten Bedingungen zur Entscheidung eines Gabelungszweigs von den Teams 31, 43 und 47 wurden aus zeitlichen Gründen nicht implementiert.

6.3.4.1 *Der optimale bzw. bessere Gabelungszweig*

Die Bestimmung des optimalen bzw. besseren Gabelungszweigs ist eine nicht-triviale Aufgabe. Beide MWB verfolgen die gleichen Ziele, aber mit konträren Prioritäten. Ein Pfad für einen beliebigen, unverzweigten Streckenabschnitt, auf dem das Objekt bewegt werden soll, sei optimal, wenn beide MWB für ihr priorisiertes Ziel den jeweils gleichen relativen Fehler machen und dieser möglichst gering ist. Auf einem geraden Streckenabschnitt z. B. entspricht die Mittellinie dem optimalen Pfad: Beide MWB machen keinen Fehler und erfüllen ihre Ziele damit zu 100%.

Für Kurven ist die Bestimmung des optimalen Pfads schwieriger. Wird stets auf der Mittellinie gefahren, entsteht ein Geschwindigkeitsfehler. Wird stets mit maximaler Geschwindigkeit gefahren, entsteht ein Genauigkeitsfehler. Während für die Geschwindigkeit ein relatives Maß für den Fehler leicht gefunden werden kann (maximale Geschwindigkeit entspricht 0%-Fehler, minimale Geschwindigkeit entspricht 100%-Fehler), ist ein gutes Maß für den Genauigkeitsfehler nicht so offensichtlich. Angenommen es wird der Abstand zur Mittellinie als Fehlermaß für die Genauigkeit genommen, wie sieht dann jeweils ein Fehler von 100% aus (insbesondere bei Kurven, die dem Streckenrand links und rechts teilweise recht Nahe sind)? Gleiches Problem tritt auf, wenn ein Flächenfehler als Fehlermaß genommen wird.

Die Bestimmung des optimalen Pfads für Gabelungen wurde für diese Arbeit als zu aufwendig erachtet. Angenommen es wird eine Methode gefunden den optimalen Pfad zu bestimmen, dann sei der optimale Gablungszweig der, bei dem der Fehler (der für beide MWB gleich sein muss) geringer ist. Bis dahin sei der schnellere Zweig als der optimale Zweig angenommen.

6.3.4.2 *Der schnellere Gabelungszweig*

Spielt die Genauigkeit keine Rolle, können die Gabelungsabschnitte mit maximaler vertikaler Geschwindigkeit durchfahren werden. Die Frage welcher Gabelungszweig schneller ist, ist dann nicht sinnvoll. Damit ein Gabelungszweig also schneller als der anderen durchfahren werden kann, muss eine Genauigkeit zu Grunde gelegt werden. Der schnellere Gabelungszweig wird daher unter der Voraussetzung bestimmt, dass das Objekt stets auf der Mittellinie[11] gefahren wird. Es wird folgende Frage beantwortet:

11 Die Wahl der Mittellinie vereinfacht die Bestimmung, da für diese bereits die Koordinaten vorliegen.

> Wenn die MWB stets mit maximal möglicher Geschwindigkeit fahren, ohne von der Mittelllinie abzukommen, wie viele Simulationsschritte benötigen sie dann jeweils für den linken und rechten Zweig der Gabelung?

Die Antwort kann in Tabelle 6.3 abgelesen werden. Während bei der abgerundeten Gabelung (RLl) beide Zweige gleich schnell durchfahren werden können, kann der linke Zweig der eckigen Gabelung neun Simulationsschritte schneller als der rechte Zweig durchfahren werden. Das entspricht einem Unterschied von $9 * 39ms = 351ms$ pro Gabelung[12].

ZWEIG	GABELUNG ELR	GABELUNG RLL
Linker Zweig	45	46
Rechter Zweig	54	46

Tabelle 6.3: Anzahl der benötigten Simulationsschritte für die Zweige der beiden Gabelungsarten.

Der Algorithmus zur Bestimmung der benötigten Simulationsschritte ist grob wie folgt aufgebaut:

1. Die aktuelle Position des Objekts wird auf das erste Koordinatenpaar der Gabelungskachel (ganz unten) gesetzt.
2. Verarbeitung des linken Gabelungszweigs:
   a) Iterativ wird ermittelt, wie weit das Objekt maximal in einem Simulationsschritt in Fahrtrichtung bewegt werden kann, ohne dass es die Mittellinie verlässt. Die ermittelte Position ist also ein Punkt auf der Mittellinie. Die ermittelte Position wird zur aktuellen Position.
   b) Die Anzahl der Iterationsschritte entspricht der benötigten Anzahl der Simulationsschritte.
   c) Ausgabe der Iterationsschritte.
3. Verarbeitung des rechten Gabelungszweigs (analog zu dem linken Gabelungszweig).

Für Details sei auf das Python-Skript `ticks_for_forks.py`[13] verwiesen. Das Script erzeugt zusätzlich Unix-Shell-Script-Dateien, die die einzelnen Simulationschritte visualisieren. Abbildung 6.8 auf Seite 78 und 6.9 auf Seite 78 zeigen die Visualisierungen. Die

12 Auf der Strecke 'hauptabschnitt_lang' sind fünf Gabelungen diesen Typs verteilt. Das entspricht einem theoretischen Unterschied von $9 * 351ms = 1755ms$ für diese Strecke.

13 Das Skript ist auf der beigelegten DVD enthalten und im Repository unter http://www.assembla.com/code/ATEO/git/nodes/master/tools/ticks_for_forks zu finden.

schwarzen Linien entsprechen der Mittellinie. Die Veränderung der Position des Objekts von einem Simulationsschritt zum nächsten wird durch eine vertikale und horizontale Linie dargestellt. Die Länge dieser Linien entspricht jeweils der absoluten Differenz einer Position zur nächsten für die X-Achse (horizontale Linie) und Y-Achse (vertikale Linie). Eine rote Linie kennzeichnet die maximale Differenz - d. h. das Objekt konnte mit maximaler Geschwindigkeit horizontal und/oder vertikal bewegt werden.

- *Visualisierung für Gabelung ELr* (Abbildung 6.8 auf der nächsten Seite)
  Auf dem linken Zweig kann auf der Geraden die maximale vertikale Geschwindigkeit gefahren werden. Auf dem Abschnitt davor und danach muss langsamer gefahren werden, aber das ist immerhin schneller als auf den entsprechenden Abschnitten der Gabelung RLl (rechter Zweig).
  Bis auf den äußersten Anfang, das äußerste Ende und auf mittlerer Höhe kann auf dem rechten Zweig zu keinem Zeitpunkt mit maximaler vertikaler Geschwindigkeit gefahren werden, sodass insgesamt der linke Zweig bei dieser Gabelung schneller durchgefahren werden kann.

- *Visualisierung für Gabelung RLl* (Abbildung 6.9 auf der nächsten Seite)
  Auf dem linken Zweig kann auf ungefähr mittlerer Höhe mit maximaler vertikaler Geschwindigkeit gefahren werden. Auf dem Abschnitt davor und danach ist es annähernd die maximale Geschwindigkeit.
  Auf dem rechten Zweig kann auf der Geraden die maximale Geschwindigkeit gefahren werden. Aber davor und danach ist die vertikale Geschwindigkeit durch die Kurven stark eingeschränkt, sodass insgesamt kein Unterschied zum linken Zweig entsteht.

6.3.4.3 *Zuordnung der geforderten Zweige zu den Gablungen*

Die geforderten Zweige (vgl. Abschnitt 4.4 auf Seite 36) und ihre Zuordnung zu den beiden Gabelungstypen sei Tabelle 6.4 entnommen.

Der dünne, breite, kurze und lange Zweig sind durch Betrachtung der beiden Gabelungen leicht ermittelbar. Der bessere bzw. optimale Zweig sei der schnellere Zweig, wie oben erwähnt. Wie ebenfalls oben erwähnt, ist keiner der beiden Zweige für die runde Gabelung (RLl) schneller. Es wurde der rechte gewählt, da dieser durch die lange Gerade zumindest schneller zu sein scheint. Als einfacher Zweig wurde jeweils der breite Zweig gewählt. Dieser sieht nicht nur wegen der Breite einfacher zu durchfahren aus, sondern fordert den MWBn auch einfachere

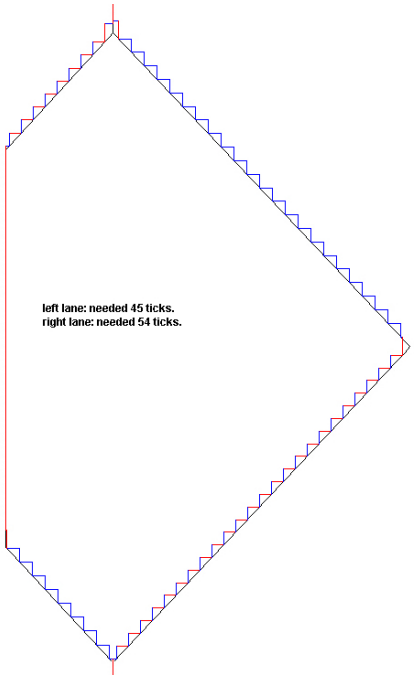


Abbildung 6.8: Bestimmung des schnelleren Gabelungszweigs (Gabelung RLl). Der linke Zweig kann schneller durchgefahren werden.

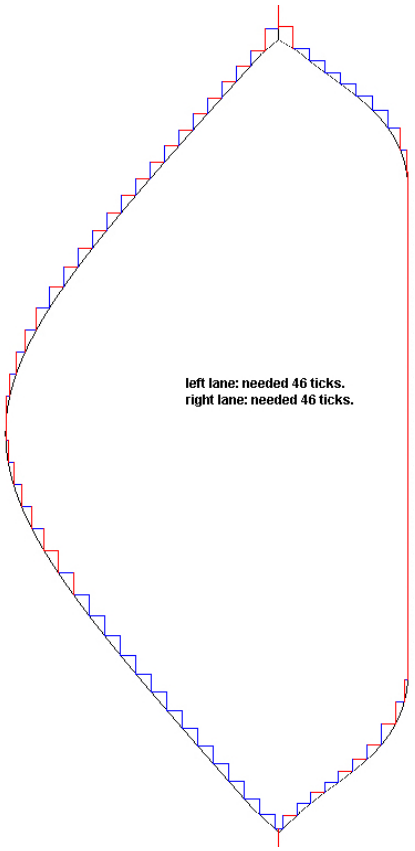


Abbildung 6.9: Bestimmung des schnelleren Gabelungszweigs (Gabelung RLl). Kein Zweig kann schneller durchgefahren werden.

Lenkbewegungen des Joysticks ab. Der Schwierige Zweig ist entsprechend das Gegenteil. Der Zufällige Zweig wird erst zufällig vor der Gabelung bestimmt und kann nicht fest angegeben werden.

ZWEIG	GABELUNG ELR	GABELUNG RLL
Optimaler Zweig	*Schneller Zweig*	*Schneller Zweig*
Dünner Zweig	Links	Rechts
Breiter Zweig	Rechts	Links
Kurzer Zweig	Links	Rechts
Langer Zweig	Rechts	Links
Schneller Zweig	Links	Rechts
Rechter Zweig	Rechts	Rechts
Einfacher Zweig	*Breiter Zweig*	*Breiter Zweig*
Schwieriger Zweig	*Gegenteil Breiter Zweig*	*Gegenteil Breiter Zweig*
Zufälliger Zweig	Links/Rechts	Links/Rechts

Tabelle 6.4: Zuordnung der geforderten Zweige zu den beiden Gabelungen.

6.3.5 *Streckenvorschau*

Die Funktion „Streckenvorschau (visuell)“ ist als dynamischer Hinweis (siehe Abschnitt 4.3.2) einzustufen, da die Anzeige für jeden Simulationsschritt angepasst wird.

Diese Funktion verwendet intern mehrere `BitBlt`-Objekte, die jeweils für den aktuellen Simulationsschritt generiert werden. Der erste Ansatz für die verkleinerte Darstellung der Strecke bestand darin, dass in der `compute:` aus der im Speicher vorliegenden Bild-Datei der relevante Ausschnitt ausgeschnitten und verkleinert angezeigt wurde. Es hat sich herausgestellt, dass die Verkleinern-Operation zu lange angedauert hat, als das die 39ms für einen Simulationsschritt eingehalten hätten werden können. Aus diesem Grund wurde wie folgt vorgegangen. Es wird bereits beim Laden der Funktion die gesamte im Speicher liegende Bild-Datei der Strecke auf die benötige Größe skaliert. Das Skalieren ist dann zwar noch immer notwendig (Parameter „Größe an Geschwindigkeit anpassen“, siehe Abschnitt C.12.8), kann aber bereits durch die initiale Skalierung bedeutend schneller ausgeführt werden.

6.4 SOFTWARE-ERGONOMISCHE BENUTZUNGSSCHNITTSTELLEN

Die Benutzungsschnittstellen für die Hinweisfunktionen, Ereignisse und die Ereignis-Konfiguration wurden nach den „Grundsätzen der Dialoggestaltung“ (vgl. 3.2.3) gestaltet.

Der *Aufgabenangemessenheit* und *Selbstbeschreibungsfähigkeit* wird z. B. durch den Vorschau-Bereich bei visuellen Hinweisfunktionen und der Möglichkeit des Abspielens einer ausgewählten Audiodatei bei auditiven Hinweisen entsprochen. Einerseits enthält der Benutzer Rückmeldungen zu seinen Einstellungen (Selbstbeschreibungsfähigkeit), andererseits werden ihm unnötige Arbeitsschritte erspart, da er seine Parameter kontrollieren kann (Effizienz-Aspekt der Aufgabenangemessenheit). Außerdem wurde auf sinnvolle Voreinstellungen geachtet und darauf, dass bei allen Parametern die Einheiten angegeben sind. Die visuellen Hinweisfunktionen sind z. B. *lernförderlich*, weil der Benutzer darauf hingewiesen wird, dass der Vorschau-Bereich auch interaktiv bedient werden kann. Via Drag and Drop kann die Position des Bildes in der Vorschau eingestellt werden, anstatt Koordinaten einzugeben. Diese Alternative zur Festlegung der Koordinaten unterstützt den Aspekt der *Individualisierbarkeit* (Mehrere Wege für eine Aufgabe). Drag and Drop kennt der Benutzer aus anderen Anwendungen, womit auch der *Erwartungskonformität* entsprochen wird. Dabei wurde darauf geachtet, dass der Benutzer die Bilder in den Vorschau-Bereichen (und bei der Ereignis-Konfiguration die Ereignisse) nicht aus ihren Bereich entfernen kann. Werden diese herausgezogen und „losgelassen“, gleiten sie zurück in ihren Bereich. Das ist eine Form der *Fehlertoleranz* (und Selbstbeschreibungsfähigkeit, da der Benutzer lernt, dass es nicht sinnvoll ist die Bilder herauszuziehen). Die Menge der angezeigten Daten wird dadurch reduziert (*Steuerbarkeit*), dass dem Benutzer die Möglichkeit gegeben wird für den Moment irrelevante Bereiche im Konfigurationsbereich einzuklappen. So ist die Ereignis-Konfiguration in einem eigenen Bereich implementiert und kann z. B. zugeklappt werden, wenn der Fokus auf den Parametern der Hinweisfunktion liegt.

7 TESTS

In diesem Kapitel werden die implementierten Tests beschrieben. Neben den automatisierten Tests für die Ereignisse (Abschnitt 7.1) und der Ereignis-Konfiguration (Abschnitt 7.2), wird die durchgeführte Untersuchung an dem Automatiken-GUI insgesamt und insbesondere an der Ereignis-Konfiguration (Abschnitt 7.4) dokumentiert. Im Abschnitt zur Untersuchung wird auch beschrieben, welche der entdeckten Probleme gelöst werden konnten.

7.1 TESTFRAMEWORK FÜR EREIGNISSE

Die Funktionstüchtigkeit der Ereignisklassen soll durch Tests sichergestellt werden. Die Ereignisse sind von den Daten des übergebenen SamStates abhängig. Sinnvolle Tests sind nur möglich, wenn den Instanzen der Ereignisklassen über mehrere Simulationsschritte realistische SamStates übergeben werden können.

Die erste Idee bestand darin, einen Agenten mit dem zu testenden Ereignis zu konfigurieren und diesen bei einem Versuchsschritt einzusetzen. Während des Versuchsschritts sollte aufgezeichnet werden, zu welchen SamStates das Ereignis eintritt und zu welchen nicht. Die gewonnenen SamStates sollten dann in einer Schleife der Reihenfolge nach dem Objekt der zu testenden Ereignisklasse übergeben werden. Ein Test ist erfolgreich, wenn das Ereignis zu den SamStates eintritt, zu denen es auch beim Aufzeichnen eingetreten ist.

Dieser Ansatz wurde verworfen. Einerseits ist die geeignete Speicherung eines SamStates aufwendig, insbesondere für komplexere Datenstrukturen wie assoziative Arrays, andererseits ist der Ablauf des Tests nicht visualisiert. Nur der Entwickler, der den Test entworfen hat, weiß wie die Strecke und Situation genau aussieht, zu der die Ereignisklasse getestet wird.

Aus diesen Gründen wurde der folgende Ansatz verfolgt. Statt die SamStates zu speichern, werden die Joystick-Eingaben abgespeichert. Dabei handelt es sich um Koordinaten, die relativ einfach als String gespeichert werden können. Werden die aufgezeichneten Koordinaten in einen Versuchssschritt eingespeist, so werden auch die gleichen SamStates generiert, die beim Aufzeichnen generiert wurden. Dadurch, dass für einen Test ein Versuchsschritt gestartet wird, steht die gewünschte Visualisierung zur Verfügung: SAM zeigt alles an, was auch beim Aufzeichnen der Joystick-Eingaben angezeigt wurde. Neben dem zeitlich aufwendigen Start eines Versuchsschritts ist dabei auch problematisch,

dass der Versuchsschritt nur manuell über ein GUI gestartet werden kann. Das erste Problem lässt sich reduzieren, das zweite vollständig lösen. Das Einbeziehen von SAM in die Tests ist sogar als Vorteil zu sehen, da hierdurch Fehler in dieser Komponente früher entdeckt werden können.

Der skizzierte Ansatz führte zur Entwicklung eines Testframeworks. In den folgenden Unterabschnitten werden für das Framework die genauen Anforderungen beschrieben, ein Beispiel demonstriert und die Umsetzung dokumentiert. Für Entwickler ist Anhang B von Interesse, da in diesem nicht nur beschrieben wird, wie Ereignisse implementiert werden können, sondern wie diese auch mit dem hier beschriebenen Framework getestet werden können.

7.1.1 *Anforderungen*

Die Anforderungen an das Testframework sowie Möglichkeiten zur Umsetzung dieser werden im Folgenden beschrieben.

1. *Nutzung des SUnit-Frameworks von Smalltalk*
   In Squeak steht das SUnit-Framework für automatisierte Unit-Tests zur Verfügung. Es ist gut dokumentiert und einfach zu benutzen. Mit dem GUI „Test Runner“ können Tests auf einfache Weise ausgeführt werden. Da das SUnit-Framework den nachfolgenden Anforderungen nicht widerspricht, soll es dem zu entwickelnden Framework zu Grunde liegen.
   Diese Anforderung kann erfüllt werden, wenn eine neue Klasse für einen Testfall eingeführt wird, die von `TestCase` erbt. Die neue Klasse erweitert die Funktionalität von `TestCase` um den folgenden Anforderungen gerecht zu werden.

2. *Automatisierte Tests für einen Versuchsschritt mit AAF*
   Tests bei denen der Entwickler eingreifen muss, sind bei mehrmaliger Ausführung sehr mühsam. Daher ist es erstrebenswert, automatisierte Tests zu ermöglichen.
   Um diese Anforderung zu erfüllen, müssen zwei Aspekte sichergestellt sein:

   a) SAM muss ohne GUI konfiguriert und gestartet werden können.

   b) Es muss möglich sein, die Joystick-Eingaben für einen Versuchsschritt aufzuzeichnen, um sie später für einen Test abzuspielen.

   Sind die beiden Aspekte erfüllt, kann ein Test ohne Benutzereingaben ablaufen.

3. *Schnell ablaufende Tests*
   Schnell ablaufende Tests sind vorteilhaft, weil sie verglichen mit langandauernden Tests häufiger ausgeführt werden und somit Fehler frühzeitiger aufdecken können.
   Dieser Anforderung wird mit folgenden Aspekten begegnet:

   a) Das Laden von Strecken nimmt die meiste Zeit beim Starten eines Versuchsschritts in Anspruch. Eine Strecke, die in einem Versuchsschritt zuvor benutzt wurde, soll nicht erneut geladen werden.

   b) Die Simulationsschrittdauer von voreingestellten 39ms wird variabel gemacht, damit diese für Testzwecke reduziert werden kann.

   c) Für einen Versuchsschritt kann festgelegt werden, wo das Objekt zu Beginn des Versuchsschritts positioniert wird. Dadurch wird es möglich, für den Test nicht relevante Streckenabschnitte zu überspringen.

4. *Lesen und Schreiben der Konfigurationsdateien*
   Ereignisse können nur verwendet werden, wenn ihre Parameter auch ordnungsgemäß in Konfigurationsdateien geschrieben und aus diesen wieder ausgelesen werden können. Es ist wichtig diesen Aspekt auch zu testen, damit die Benutzer des Automatiken-GUI keine Fehler-Popups beim abspeichern ihrer Automatiken erhalten. Jeder Test einer Ereignisklasse soll daher diese Anforderung erfüllen.

5. *Speicherung von Testergebnissen*
   Zur Nachvollziehbarkeit dieser Arbeit sollen die Testergebnisse in Dateien geschrieben werden.

6. *Manuelle Tests möglich*
   Deckt ein Testfall einen eingeführten Fehler auf, so sollen die Entwickler auf eine möglichst einfache Weise den Fehler nachvollziehen können.
   Dieser Anforderung wird mit folgenden Aspekten begegnet:

   a) Der Vergleich von Ist- und Soll-Ausgaben soll einfach möglich sein. Hierbei erweisen sich die abgespeicherten Testergebnisse als nützlich.

   b) Aufgezeichnete Joystick-Eingaben sollen deaktiviert werden können, um manuell zu testen.

Diese Anforderungen wurden umgesetzt. Zum Verständnis der nachfolgenden Abschnitte wird dem Leser empfohlen die Abschnitte zur Benutzung des Testframeworks in Anhang B zu lesen.

7.1.2 *Beispiel einer Testmethode*

In diesem Abschnitt wird die Benutzung des Testframeworks zusammenfassend anhand einer Testmethode wiedergegeben. Für Details sei auf Anhang B verwiesen.

Aus der Klasse `AAFTrackObstacleAheadTest` wird die Testmethode `testDynamicObstacle` exemplarisch vorgestellt. Mit dieser Testmethode wird die Erkennung der dynamischen Hindernisse sichergestellt.

Listing 7.1: Tetstmethode testDynamicObstacle

```
testDynamicObstacle
  "self debug: #testDynamicObstacle"

  | event |
  "CONFIGURE EVENT"
  event := AAFTrackObstacleAhead new.
  event lowerBoundOffset: -800 upperBoundOffset: 100.
  event typesOfInterest: { #nextDynamicObstacle }
      asOrderedCollection.

  "SETUP AGENT"
  self setupAgents:
    (Array
      with: (AAFInputProviderAgent withJoystickInputs: (self
          class testInput))
      with: (AAFTestAgent withDelegate: event)
    ).

  "SETUP STEP"
  self configureStepMwiControl: '1'.
  self configureStepTrack: 'hauptabschnitt2'.
  self configureStepObstacleConfig: 'obstacleConfig_2'.
  self configureStepStartAtDistance: 8500.
  self configureStepEndAtDistance: 26100.
  self configureStepTickDuration: #( #(0 99999999 5)).

  "RUN STEP"
  self runStep.

  "PRINT & COMPARE OUTPUT"
  self printAndCompareOutputWith: self class
      testOutputReferenceDynamicObstacle.
```

Der Quelltext der Testmethode folgt grob einer festen Struktur. Die verschiedenen Abschnitte werden durch großgeschriebene Kommentare eingeteilt:

- *CONFIGURE EVENT*
  Das zu testende Ereignis wird konfiguriert. Dazu wird eine Instanz der Ereignisklasse `AAFTrackObstacleAhead` erstellt und in den nachfolgenden Zeilen konfiguriert.

- *SETUP AGENT*
  In diesem Abschnitt werden die Hilfsagenten konfiguriert, die den Test ermöglichen. Der erste Agent ist vom Typ `AAFInputProviderAgent` und simuliert aufgezeichnete Joystick-Eingaben.
  Der zweite Agent vom Typ `AAFTestAgent` nimmt das konfigurierte Ereignis entgegen und integriert es in den Versuchsschritt.

- *SETUP STEP*
  Der SAM-Versuchsschritt wird konfiguriert.

- *RUN STEP*
  Der Versuchsschritt wird gestartet und läuft mit dem konfigurierten Ereignis ab.

- *PRINT & COMPARE OUTPUT*
  Die generierte Ist-Ausgabe des Ereignisses wird mit einer Soll-Ausgabe verglichen.

Die einzelnen Methoden des Testframeworks werden in Abschnitt B.3.5 beschrieben.

Die gezeigte Testmethode kann über das Tool TestRunner vom SUnit-Framework ausgeführt werden oder indem der Benutzer den Kommentar `self debug: #testDynamicObstacle` ohne Anführungszeichen markiert und mit der Tastenkombinaton Strg+D ausführt. Bei Abweichung der Ist-Ausgabe von der Soll-Ausgabe öffnet sich nach Ablauf des Tests ein Debug-Fenster.

Die Joystick-Eingaben sind in der Klassenmethode `testInput` als String gespeichert, der von der Methode zurückgegeben wird:

Listing 7.2: Joystick-Eingabedaten in AAFTrackObstacleAheadTest

```
AAFTrackObstacleAheadTest testInput

  "super lastJoystickInputs."

  ^
'527@0 0@0
664@0 0@0
664@-85 0@0
495@-85 0@0
442@-107 0@0
'
```

Dabei handelt es sich um zwei Koordinatenpaare. Das erste Paar ist vom Joystick des ersten MWBs, das zweite Paar vom Joystick des zweiten MWBs generiert.

Die Soll-Ausgabe des Ereignisses wird ebenfalls als String einer Klassenmethode zurückgegeben:

Listing 7.3: Soll-Ausgabe in AAFTrackObstacleAhead

```
AAFTrackObstacleAheadTest testOutputReference

  ^
'9073: obstacle nextDynamicObstacle
9084: obstacle nextDynamicObstacle
9094: obstacle nextDynamicObstacle
9893: obstacle nextDynamicObstacle
'
```

Die Zahl vor dem Doppelpunkt ist die zurückgelegte Anzahl von Pixeln (entspricht `totalStepDistance` des SamStates). Die Zeichenkette danach ist eine zu der Distanz (und damit auch zu einem Simulationsschritt) generierte Ausgabe des Ereignisses. In diesem Fall wird der Hindernistyp ausgegeben, zu dem das Ereignis eintritt.

Der folgende Abschnitt beschreibt die wesentlichen Klassen des Testframeworks und dokumentiert damit dessen Umsetzung.

7.1.3 *Umsetzung*

Das Testframework wurde mit drei Klassen umgesetzt (siehe Abbildung 7.1):

AAFEVENTTESTCASE Diese Klasse stellt die notwendige Funktionalität zur Verfügung, um einen Versuchsschritt mit Agenten zu konfigurieren und zu starten. Weil die Klasse von `TestCase` ableitet, stellt sie all die Funktionalität zur Verfügung, die von SUnit-Testklassen bekannt sind (insbesondere die Integration in den *Test Runner*). Nach Ablauf des Versuchsschritts können die (Ist-)Ausgaben des gekapselten Ereignis-Objekts (siehe nächste Klasse) mit den Soll-Ausgaben verglichen werden. Des Weiteren enthält die Klasse Methoden, um auf die Joystick-Daten und Ausgaben des letzten Versuchsschritts zuzugreifen, um einen neuen automatisierten Test zu erstellen.

AAFTESTAGENT Diese Klasse ist ein AAFAgent. Sie nimmt ein Ereignis-Objekt und integriert es in den Ablauf des Versuchsschritts.

AAFINPUTPROVIDERTEST Diese Klasse speist während des Versuchsschritts aufgenommene Joystick-Daten in SAMModel-Data ein und simuliert damit Joystick-Eingaben. Alternativ können Joystick-Eingaben generiert werden die einem ruhenden Joystick (keine Auslenkung) entsprechen. Durch diese Klasse werden automatisierte Tests ermöglicht.

Abbildung 7.1: Klassendiagramm des Testframeworks

Zur Benutzung dieser Klassen und insbesondere der Methoden sei auf Anhang B verwiesen. Nachfolgend werden die Tests beschrieben.

7.2 TESTS DER EREIGNISSE

In den nachfolgenden Abschnitten werden die automatisierten Tests für alle implementierten Ereignisse dokumentiert.

7.2.1 *Ereignis „Bevor/im Gabelungsbereich“*

Die Klasse `AAFTrackForkAheadTest` enthält Testmethoden für das Ereignis „Bevor/im Gabelungsbereich“.

7.2.1.1 *Strecke und Hinderniskonfiguration*

Als Strecke wurde „hauptabschnitt2“ gewählt, da diese Strecke beide Gabelungstypen enthält. Die Reihenfolge der Gabelungen auf dieser Strecke ist:

1. Eckige Gabelung

2. Runde Gabelung
3. Runde Gabelung

Weil eine Gabelung doppelt vorkommt, wird auch getestet, ob das Ereignis auf der Strecke mehrmals für einen Gabelungstypen (runde Gabelung) eintreten kann.

Es wurde die Hinderniskonfiguration „obstacleConfig_0“ gewählt, um keine störenden Hindernisse auf der Strecke zu platzieren.

7.2.1.2 *Testmethoden*

- Methode `testBothForks`
  Testet die Erkennung von runden und eckigen Gabelungstypen.
- Methode `testCorneredFork`
  Testet die Erkennung von eckigen Gabelungen, runde Gabelungen sollen nicht erkannt werden.
- Methode `testRoundFork`
  Testet die Erkennung von runden Gabelungen, eckige Gabelungen sollen nicht erkannt werden.
- Methode `testWarningBeforyAnyFork`
  Testet, ob das Ereignis ein- und austreten kann, bevor die Gabelung passiert wird (für Warnungen vor Gabelungen).

7.2.2 *Ereignis „Bevor/im Hindernisbereich“*

Die Klasse `AAFTrackObstacleAheadTest` enthält Testmethoden für das Ereignis „Bevor/im Hindernisbereich“.

7.2.2.1 *Strecke und Hinderniskonfiguration*

Als Strecke wurde „hauptabschnitt2“ mit der Hinderniskonfiguration „obstacleConfig_2“ gewählt, da bei dieser Zusammenstellung alle Hindernistypen getestet werden können. Die Reihenfolge der Hindernisse auf dieser Strecke bei der genannten Hinderniskonfiguration ist:

1. Ein dynamisches Hindernis
2. Zwei statische Hindernisse mit 25%-Abdeckung der Fahrbahn. Das erste Hindernis ist auf der linken, das zweite auf der rechten Fahrspur (Slalom).
3. Zwei statische Hindernisse mit 50%-Abdeckung der Fahrbahn. Das erste Hindernis ist auf der linken, das zweite auf der rechten Fahrspur (Slalom).

7.2.2.2 *Testmethoden*

Für die Testmethoden wurden Joystick-Eingaben aufgezeichnet, die beim Aufrufen der Testmethoden eingespielt werden. Vor den statischen Hindernissen mit 50%-Abdeckung der Fahrbahn wird ausgewichen.

- Methode `testObstacle`
  Testet die Erkennung von einem beliebigen Hindernis. Jeder Hindernistyp soll erkannt werden.

- Methode `testDynamicObstacle`
  Testet die Erkennung von einem dynamischen Hindernis. Alle anderen Hindernisse sollen nicht erkannt werden.

- Methode `testStaticObstacle25Left`
  Testet die Erkennung von einem statischen Hindernis mit 25%-Abdeckung der Fahrbahn, welches sich auf der linken Fahrspur befindet. Alle anderen Hindernisse sollen nicht erkannt werden.

- Methode `testStaticObstacle25Right`
  Testet die Erkennung von einem statischen Hindernis mit 25%-Abdeckung der Fahrbahn, welches sich auf der rechten Fahrspur befindet. Alle anderen Hindernisse sollen nicht erkannt werden.

- Methode `testStaticObstacle50Left`
  Testet die Erkennung von einem statischen Hindernis mit 50%-Abdeckung der Fahrbahn, welches sich auf der linken Fahrspur befindet. Alle anderen Hindernisse sollen nicht erkannt werden.

- Methode `testStaticObstacle50Right`
  Testet die Erkennung von einem statischen Hindernis mit 50%-Abdeckung der Fahrbahn, welches sich auf der rechten Fahrspur befindet. Alle anderen Hindernisse sollen nicht erkannt werden.

- Methode `testSlalom25`
  Testet die Erkennung von einem Hindernis-Slalom mit 25%-Fahrbahnabdeckung. Die beiden einzelnen Hindernisse werden wie ein großes Hindernis angesehen. Alle anderen Hindernisse sollen nicht erkannt werden.

- Methode `testSlalom50`
  Testet die Erkennung von einem Hindernis-Slalom mit 50%-Fahrbahnabdeckung. Die beiden einzelnen Hindernisse werden wie ein großes Hindernis angesehen. Alle anderen Hindernisse sollen nicht erkannt werden.

- Methode `testSlalom`
  Testet die Erkennung von Hindernis-Slaloms mit 25%- und 50%-Fahrbahnabdeckung. Alle anderen Hindernisse sollen nicht erkannt werden.

7.2.3 *Ereignis „Bevor/im Kurvenbereich“*

Die Klasse `AAFTrackCurveAheadTest` enthält Testmethoden für das Ereignis „Bevor/im Kurvenbereich“.

7.2.3.1 *Strecke und Hinderniskonfiguration*

Als Strecke wurde „hauptabschnitt2“ gewählt, da diese Strecke alle Kurvenarten enthält. Es wurde die Hinderniskonfiguration „obstacleConfig_0“ gewählt, um keine störenden Hindernisse auf der Strecke zu platzieren.

7.2.3.2 *Testmethoden*

- Methode `testAllCurves`
  Testet die Erkennung von einer beliebigen Kurve. Jeder Kurventyp soll erkannt werden.
- Methode `testLeftCurve30`
  Testet die Erkennung von einer Links-Kurve der Breite 30px.
- Methode `testLeftCurve90`
  Testet die Erkennung von einer Links-Kurve der Breite 90px.
- Methode `testLeftCurve150`
  Testet die Erkennung von einer Links-Kurve der Breite 150px.
- Methode `testLeftCurve300`
  Testet die Erkennung von einer Links-Kurve der Breite 300px.
- Methode `testRightCurve30`
  Testet die Erkennung von einer Rechts-Kurve der Breite 30px.
- Methode `testRightCurve90`
  Testet die Erkennung von einer Rechts-Kurve der Breite 90px.
- Methode `testRightCurve150`
  Testet die Erkennung von einer Rechts-Kurve der Breite 150px.
- Methode `testRightCurve300`
  Testet die Erkennung von einer Rechts-Kurve der Breite 300px.

7.2.4 *Ereignis „Bremsempfehlung (dyn. Hindernis)“*

Die Klasse `AAFDrivingDynObsBrakeNowTest` enthält Testmethoden für das Ereignis Bremsempfehlung (dyn. Hindernis)“.

Beide Testmethoden machen von unterschiedlichen Joystick-Eingaben Gebrauch, um den gewünschten Test zu ermöglichen.

7.2.4.1 *Testmethoden*

- Methode `testDangerOfCollision`
  Durch Fahren auf Kollisionskurs wird getestet, ob eine Bremsempfehlung signalisiert wird.
- Methode `testNoDangerOfCollision`
  Durch Meiden des Kollisionskurses wird getestet, ob fälschlicherweise eine Bremsempfehlung signalisiert wird.

7.2.5 *Ereignis „Beschleunigungsempfehlung (dyn. Hindernis)“*

Die Klasse `AAFDrivingDynObsSpeedUpNowTest` enthält Testmethoden für das Ereignis „Beschleunigungsempfehlung (dyn. Hindernis)“.

Beide Testmethoden machen von unterschiedlichen Joystick-Eingaben Gebrauch, um den gewünschten Test zu ermöglichen.

7.2.5.1 *Testmethoden*

- Methode `testDangerOfCollision`
  Durch Fahren auf Kollisionskurs wird getestet, ob eine Beschleunigungsempfehlung signalisiert wird.
- Methode `testNoDangerOfCollision`
  Durch Meiden des Kollisionskurses wird getestet, ob fälschlicherweise eine Beschleunigungsempfehlung signalisiert wird.

7.2.6 *Ereignis „Zu schnell fahren“*

Die Klasse `AAFDrivingTooFastTest` enthält Testmethoden für das Ereignis „Zu schnell fahren“.

7.2.6.1 *Testmethoden*

- Methode `testIt`
  Durch Variation der Geschwindigkeit wird getestet, ob das Ereignis korrekt ein- und austritt.

7.2.7 *Ereignis „Zu langsam fahren“*

Die Klasse `AAFDrivingTooSlowTest` enthält Testmethoden für das Ereignis „Zu langsam fahren“.

7.2.7.1 *Testmethoden*

- Methode `testIt`
  Durch Variation der Geschwindigkeit wird getestet, ob das Ereignis korrekt ein- und austritt.

7.2.8 *Ereignis „Auf der Fahrbahn“*

Die Klasse `AAFDrivingOnTrackTest` enthält Testmethoden für das Ereignis „Auf der Fahrbahn“.

7.2.8.1 *Testmethoden*

- Methode `testIt`
  Durch unterschiedliche Positionierung des Objekts auf und abseits der Fahrbahn wird getestet, ob das Ereignis korrekt ein- und austritt.

7.2.9 *Ereignis „Eingabe-Timeout“*

Die Klasse `AAFDrivingWithoutInputTest` enthält Testmethoden für das Ereignis „Eingabe-Timeout“.

7.2.9.1 *Testmethoden*

- Methode `testIt`
  Durch die Joystick-Ruhestellung wird getestet, ob das Ereignis korrekt ein- und austritt.

7.2.10 *Ereignis „Auf einer Seite der Mittellinie“*

Die Klasse `AAFDrivingOneSideOfRacingLineTest` enthält Testmethoden für das Ereignis „Auf einer Seite der Mittellinie“.

7.2.10.1 *Testmethoden*

- Methode `testIt`
  Durch unterschiedliche Positionierung des Objekts links und rechts von der Mittellinie wird getestet, ob das Ereignis korrekt ein- und austritt.

7.3 TESTS DER EREIGNIS-KONFIGURATION

In diesem Abschnitt wird auf die Tests der Ereignis-Konfiguration eingegangen.

7.3.1 *Manuelle Tests*

Mit dem Aufruf `AAFEventConfigurator openInWindowDebug` wird die Ereignis-Konfiguration in einem eigenem Fenster gestartet (unabhängig des Automatiken-GUI) und alle verfügbaren Ereignisse werden in diesem GUI angezeigt. Es handelt sich dabei um keinen Testfall im eigentlichen Sinne, sondern dient nur dem Entwickler, der seine Änderungen am Quelltext überprüfen kann, ohne auf das Starten des Automatiken-GUI zu warten.

7.3.2 *Automatische Tests*

Die Klasse `AAFEventConfiguratorTest` enthält Testmethoden, die Ereignis-Bäume einlesen und die entsprechende Konfiguration in der GUI anzeigen. Der eigentliche Vergleich läuft darauf hinaus, ob die erwarteten GUI-Widgets generiert und korrekt verschachtelt sind.

7.3.2.1 *Testmethoden*

- Methode `testInitFromEventTree1`
  Testet einen Ereignis-Baum mit einem Ereignis.
- Methode `testInitFromEventTree2`
  Testet einen Ereignis-Baum mit einer UND-Verknüpfung und zwei Ereignissen.
- Methode `testInitFromEventTree3`
  Testet einen Ereignis-Baum mit einer ODER- sowie UND-Verknüpfung und drei Ereignissen (entsprechend geschachtelt).
- Methode `testInitFromEventTree4`
  Testet einen Ereignis-Baum mit einer UND- sowie zwei ODER-Verknüpfung und vier Ereignissen (entsprechend geschachtelt).

7.4 UNTERSUCHUNG DES AUTOMATIKEN-GUI UND DER EREIGNIS-KONFIGURATION

Die Verständlichkeit und Gebrauchstauglichkeit des Automatiken-GUI und insbesondere der Ereignis-Konfiguration wurde durch

eine kleine Usability-Untersuchung festgestellt. Das Automatiken-GUI wurde bereits im Rahmen der Studienarbeit von Kosjar (2011) bezüglich der Gebrauchstauglichkeit aufgebessert. Allerdings wurde bisher noch kein Test mit projekt-externen Benutzern durchgeführt. Das wird an dieser Stelle nachgeholt. Die Untersuchung konnte im Rahmen dieser Arbeit nur mit vier Versuchspersonen durchgeführt werden. Auch wenn 32 Probleme festgestellten werden konnten, ist die Untersuchung als ein Vortest anzusehen. Eine umfangreichere Untersuchung wird empfohlen.

7.4.1 *Allgemeines zur Untersuchung*

Die durchgeführte Untersuchung ist grob wie folgt charakterisiert.

- *Versuchspersonen* (siehe Abschnitt 7.4.3)
  Die Untersuchung wurde mit vier projekt-externen Versuchspersonen durchgeführt.
- *Ort, Zeit und Dauer der Untersuchung*
  Die Untersuchung wurde an zwei Tagen von 17 bis 19 Uhr in den Räumlichkeiten des Instituts für Psychologie der Humboldt-Universität zu Berlin durchgeführt. Mit zwei Vorversuchen wurde ermittelt, dass eine Untersuchung ungefähr eine Stunde dauerte.
- *Grober Ablauf der Untersuchung* (siehe Abschnitt 7.4.2)
  Die Versuchspersonen studierten zuerst das Automatiken-GUI mit Hilfe der Bedienungsanleitung, bearbeiteten anschließend vier Aufgaben und beantworteten schließlich noch Fragen in einem Interview. Während der Untersuchung haben die Versuchspersonen laut gedacht. Die gesamte Untersuchung wurde vom Verfasser geleitet.
- *Unterlagen zur Versuchsdurchführung*
  Die Versuchspersonen bekamen folgende Unterlagen:
  - *Vier A5-Blätter mit den Versuchsaufgaben*
    Auf jedem Blatt stand eine Aufgabe. Die Blätter wurden den Versuchspersonen nacheinander, jeweils vor der Bearbeitung der speziellen Aufgabe, vorgelegt. Die Aufgaben werden in Abschnitt 7.4.4 vorgestellt.
  - *Ein A4-Blatt mit zwei Bildern von SAM*
    Abbildung 7.2 zeigt die beiden Bilder. Das A4-Blatt lag während der gesamten Untersuchung neben dem Laptop und sollte den Versuchspersonen den Kontext vor Augen halten.

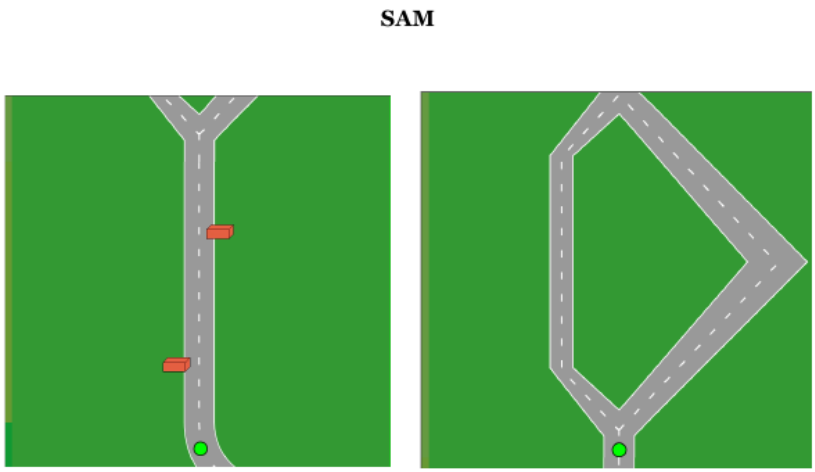


Abbildung 7.2: Die beiden Bilder von SAM sollten den Versuchspersonen den Kontext vor Augen halten.

- *Geräte und Programme zur Versuchsdurchführung*
  Die Untersuchung wurde an einem bereitgestellten Laptop des Instituts für Psychologie durchgeführt. Während der Untersuchung liefen auf dem Laptop folgende Programme:
  - *Adobe Reader mit geöffneter Bedienungsanleitung des Automatiken-GUIs*
  - *Squeak mit einem geöffneten Autmatiken-GUI im maximierten Fenster*
    Das verwendete Automatiken-GUI ist vom 12. Februar 2012[1]. Das Squeak-Image war so konfiguriert, dass die folgenden elementaren Funktionen im Automatiken-GUI zur Verfügung standen: Adaptives Lenken, Bobbahn, Einflussregulierung, Einflussverteilung, Eingabeumkehr, Gabelungshinweis (auditiv), Gabelungshinweis (visuell), Hinweis (auditiv), Hinweis (visuell) und Streckenvorschau. Es standen keine kombinierten Funktionen zur Verfügung, da diese erst durch das Bearbeiten der Aufgaben erstellt werden sollten.
  - *Morae von TechSmith als „Usability Testing Tool" (lizenziert für das Institut für Psychologie)*
    Diese Anwendung lief im Hintergrund und wurde daher von den Versuchspersonen nicht wahrgenommen. Durch das Mikrofon und die Kamera am Laptop wurden die Stimme und das Gesichtsfeld der Versuchspersonen aufgezeichnet. Zusätzlich zeichnete Morae alles auf, was das Display des Laptops während der Untersuchung angezeigt hat.
    Diese Aufzeichnungen dienen der Auswertung sowie

1 In der Versionsverwaltung des Projekts entspricht diese Version dem Git-Hash 62846db87722a8b1efb613634e2edf01ccfdbf8a.

Nachvollziehbarkeit der Untersuchung. Die aufgenommenen Videos und Morae-Projektdateien sind Teil dieser Arbeit und befinden sich auf der beigelegten DVD (siehe Anhang D).

7.4.2 *Ablauf einer Untersuchung*

Der Ablauf der Untersuchung besteht aus vier Schritten:

1. *Begrüßung und kurze Vorstellung des Projekts sowie des Versuchsablaufs* (ca. 3 Minuten)
   Den Versuchspersonen wurde SAM kurz gezeigt und als kooperative Tracking-Simulation erklärt. Das Automatiken-GUI soll auf Gebrauchstauglichkeit getestet werden. Es dient der Aktivierung und Konfiguration von Automatiken bzw. Assistenzsystemen, welche die MWB unterstützen.

2. *Studium der Bedienungsanleitung und des Automatiken-GUI (ca. 25 Minuten)*
   Am Laptop bekamen die Versuchspersonen das Automatiken-GUI und die zugehörige Bedienungsanleitung gezeigt. Sie haben ungefähr eine halbe Stunde zur Einarbeitung gehabt. Die Versuchspersonen wurden ermutigt, das Gelesene am geöffneten Automatiken-GUI nachzuvollziehen.

3. *Bearbeitung der Aufgaben (ca. 20 Minuten, siehe 7.4.4)*
   Insgesamt sollten vier aufeinander aufbauende Aufgaben bearbeitet werden. Die Aufgaben sind so gestellt, dass die Grundfunktionalität des Automatiken-GUIs und der Ereignis-Konfiguration abgedeckt werden. Die Versuchspersonen bekamen die Aufgaben nacheinander als A5-Kärtchen zur Bearbeitung vorgelegt.

4. *Interview (ca. 10 Minuten, siehe 7.4.5)*
   Durch das Interview wurde den Versuchspersonen die Gelegenheit gegeben, generelle Eindrücke des Automatiken-GUIs und der Ereignis-Konfiguration zu schildern.

Die Dauer jedes Schritts wurde anhand von zwei Vorversuchen ermittelt.

7.4.3 *Versuchspersonen*

An der Untersuchung haben vier Personen teilgenommen. Diese wurden mit Versuchspersonenmarken oder 8 EUR entlohnt. Sie haben alle gemeinsam, dass sie beruflich am Computer arbeiten, sodass die PC-Nutzung in Stunden pro Woche recht hoch ausfällt. Ihnen allen ist gemeinsam, dass sie mit herkömmlicher Büro-Software vertraut sind und am Institut für Psychologie angestellt

sind. Keine der Versuchspersonen ist mit dem ATEO-Projekt vertraut.

Tabelle 7.1 listet die Merkmale der teilgenommenen Versuchspersonen auf.

VP	GESCHLECHT	ALTER	PC-NUTZUNG	STATUS
1	weiblich	27	40h/Woche	Wiss. MA
2	männlich	27	60h/Woche	Wiss. MA
3	weiblich	46	35h/Woche	Student
4	weiblich	23	38h/Woche	Student

Tabelle 7.1: Charakteristika der vier Versuchspersonen

7.4.4 *Aufgaben*

Es wurden vier aufeinander aufbauende Aufgaben entwickelt. Jede Aufgabe verlangt die Benutzung eines neuen Aspekts des Automatiken-GUI ab. Die letzte Aufgabe dient dem eigentlichen Test der Ereignis-Konfiguration.

Die Aufgaben sind so formuliert, dass sie einen Zielzustand beschreiben anstatt präzise Handlungsanweisungen zu geben. Dadurch wird das WIE auf den Benutzer übertragen, sodass der Test auch sinnvoll ist.

7.4.4.1 *1. Aufgabe*

Erstellen Sie eine Automatik, die den Mikroweltbewohnern eine Übersicht der kommenden Streckenelemente zur Verfügung stellt und bei Gabelungen einen visuellen Richtungsvorschlag unterbreitet. Die Funktionen sollten in Folge geschaltet werden.

Speichern Sie die erstellte Automatik als „aufgabe1.aaf" ab.

Diese Aufgabe dient als einfacher Einstieg. Es wird folgende Funktionalität getestet:

1. *Elementare Funktionen aus dem Funktionsbereich in den Ablaufplan per Drag and Drop ziehen.*
   Die Aufgabenstellung umschreibt die beiden elementaren Funktionen „Streckenvorschau" und „Gabelungshinweis". Diese sollten aus insgesamt zehn elementaren Funktionen ausgewählt werden.

2. *Funktionen im Ablaufplan sequentiell verbinden.*
   Das gewünschte Ergebnis ist in Abbildung 7.3 zu sehen.

3. *Abspeichern des Ablaufplans.*

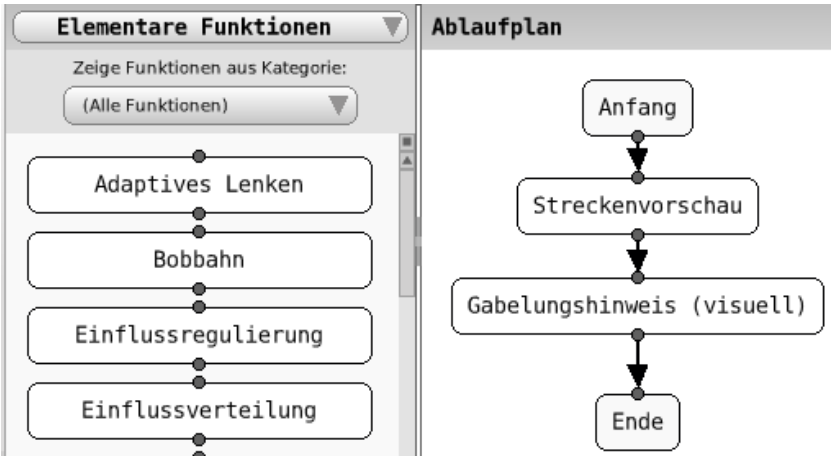


Abbildung 7.3: Erwartetes Ergebnis nach der Bearbeitung von Aufgabe 1. Auf der linken Seite sind die anderen Funktionen zu erkennen.

7.4.4.2 2. *Aufgabe*

Erstellen Sie eine weitere Automatik, die die Mikroweltbewohner nicht von der Fahrbahn abkommen lässt und die aus Aufgabe 1. abgespeicherte Automatik bzw. Funktion enthält. Die Funktionen sollten parallel geschalten werden.

Speichern Sie die erstellte Automatik als „aufgabe2.aaf" ab.

Abbildung 7.4 zeigt das erwartete Ergebnis. Es wird folgende Funktionalität getestet:

1. *Neue Automatik erstellen.*
   Dazu soll Button „Neu" in der Menüleiste benutzt werden.
2. *Umschaltung des Funktionsbereiches zur Anzeige von kombinierten Funktionen.*
   Dies wird mit Hilfe des Dropdown-Menüs gemacht. Dieses ist in Abbildung 7.4 links oben zu sehen.
3. *Gespeicherte Funktion (aus Aufgabe 1) in die neue Automatik integrieren.*
   Siehe Abbildung 7.4.
4. *Funktionen im Ablaufplan parallel verbinden.*
5. *(Abspeichern des Ablaufplans)*

7.4.4.3 3. *Aufgabe*

Verändern Sie die Automatik aus Aufgabe 2 dahingehend, dass Sie die Funktion für die Übersicht der Streckenelemente wie folgt konfigurieren:

1. *Alle Objekte bis auf die Hindernisse sollen angezeigt werden.*

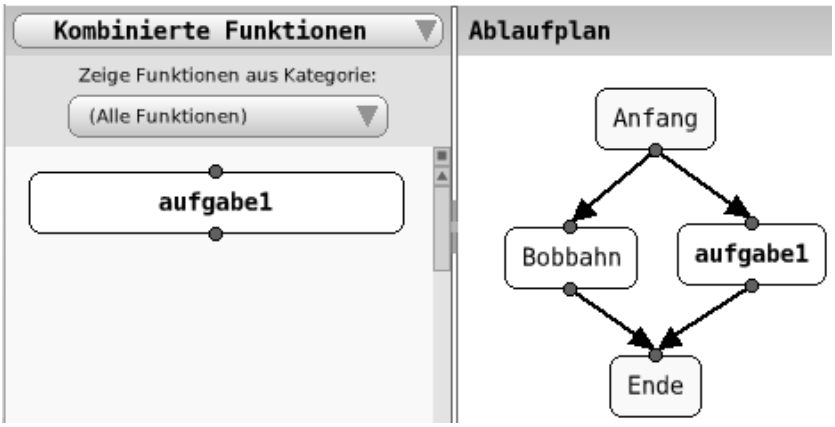


Abbildung 7.4: Erwartetes Ergebnis nach der Bearbeitung von Aufgabe 2. Das Dropdown-Menü zur Umschaltung der Funktionsbereiche ist links oben zu sehen.

2. *Die gesamte Streckenvorschau soll halb-transparent angezeigt werden.*

Speichern Sie die veränderte Automatik als „aufgabe3.aaf“ ab.

Es wird folgende Funktionalität getestet:

1. *Navigation in eine kombinierte Automatik.*
   Hierzu soll der Button „Detailgrad erhöhen“ rechts oben im Konfigurationsbereich benutzt werden. Siehe Abbildung 7.5.

2. *Ändern von Parametern zweier Funktionen.*
   Die eigentlichen Parameter sind von keiner Bedeutung. Wichtig ist, dass die Versuchspersonen den ganzen Konfigurationsbereich wahrnehmen.

3. *Navigation zurück aus einer kombinierten Funktion.*
   Hierzu soll der Button „Detailgrad verringern“ rechts oben im Konfigurationsbereich benutzt werden. Siehe Abbildung 7.6.

4. *Änderungen an einer Automatik unter einem neuen Namen abspeichern.*
   Button „Speichern unter“ statt „Speichern“ in der Menüleiste.

7.4.4.4 4. Aufgabe

Verändern Sie die Automatik aus Aufgabe 3:

1. *Die Funktion für den Richtungsvorschlag bei Gabelungen soll nur aktiv sein, wenn sich die Mikroweltbewohner vor einer Gabelung befinden und in gegensätzliche Richtungen lenken.*

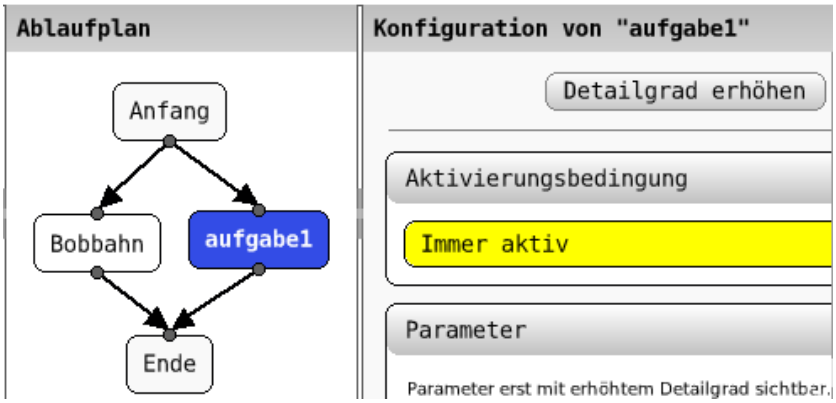


Abbildung 7.5: Erwartetes Ergebnis nach der Bearbeitung von Aufgabe 3. Rechts oben ist der Button „Detailgrad erhöhen“ zum Navigieren in den Ablaufplan von der kombinierten Funktion „aufgabe1“ zu sehen.

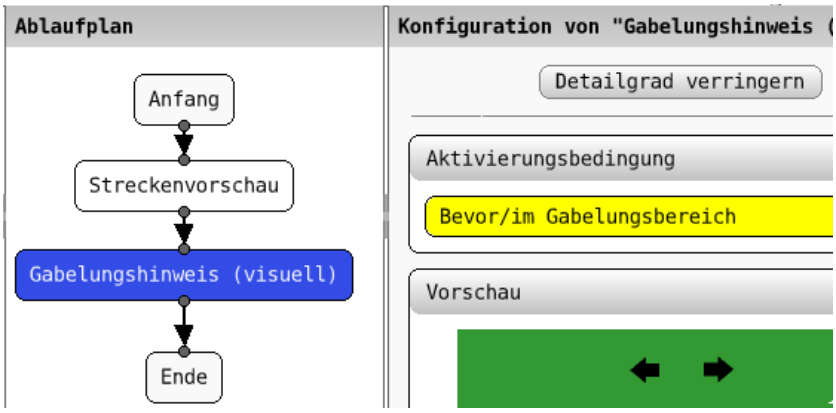


Abbildung 7.6: Aufgabe 3 - Rechts oben ist der Button „Detailgrad verringern“ zum Navigieren in den äußeren Ablaufplan zu sehen.

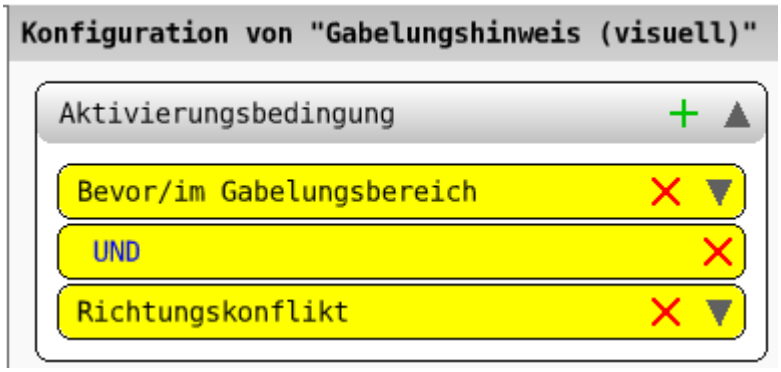


Abbildung 7.7: Erwartetes Ergebnis nach der Bearbeitung von Teilaufgabe 1 der Aufgabe 4.

2. *Die Funktion, die die Mikroweltbewohner am Verlassen der Fahrbahn hindert, soll nur aktiv sein, wenn die Mikroweltbewohner vor langen Kurven sind oder die Mikroweltbewohner vor einer Gabelung in entgegengesetzte Richtungen lenken.*

Speichern Sie die veränderte Automatik als „aufgabe4.aaf“ ab.

Es wird folgende Funktionalität getestet.

Das erwartete Ergebnis von Teilaufgabe 1 ist in Abbildung 7.7 dargestellt. Diese Teilaufgabe testet folgende Funktionalität:

1. *Operator ohne Klammern der Aktivierungsbedingung hinzufügen.*
   Hierzu muss durch ein Links-Klick auf das grüne Plussymbol das Menü geöffnet werden und der Operator „UND“ hinzugefügt werden.

2. *Ereignisse der Aktivierungsbedingung hinzufügen.*
   Wie Schritt eins, aber das Ereignis „Richtungskonflikt“ soll hinzugefügt werden.

Das erwartete Ergebnis von Teilaufgabe 2 ist in Abbildung 7.8 dargestellt. Diese Teilaufgabe testet folgende Funktionalität:

1. *Ereignisse löschen.*
   Das voreingestellte Ereignis „Immer aktiv“ soll durch klicken auf das rote X gelöscht werden.

2. *Operator mit Klammern der Aktivierungsbedingung hinzufügen.*
   Der Operator „(UND)“ soll aus dem Menü hinzugefügt werden.

3. *Ereignisse konfigurieren.*
   Das Ereignis „Bevor/im Kurvenbereich“ muss aufgeklappt und konfiguriert werden.

4. *Ereignisse und Operanden zu einem gültigen Ausdruck anordnen.*

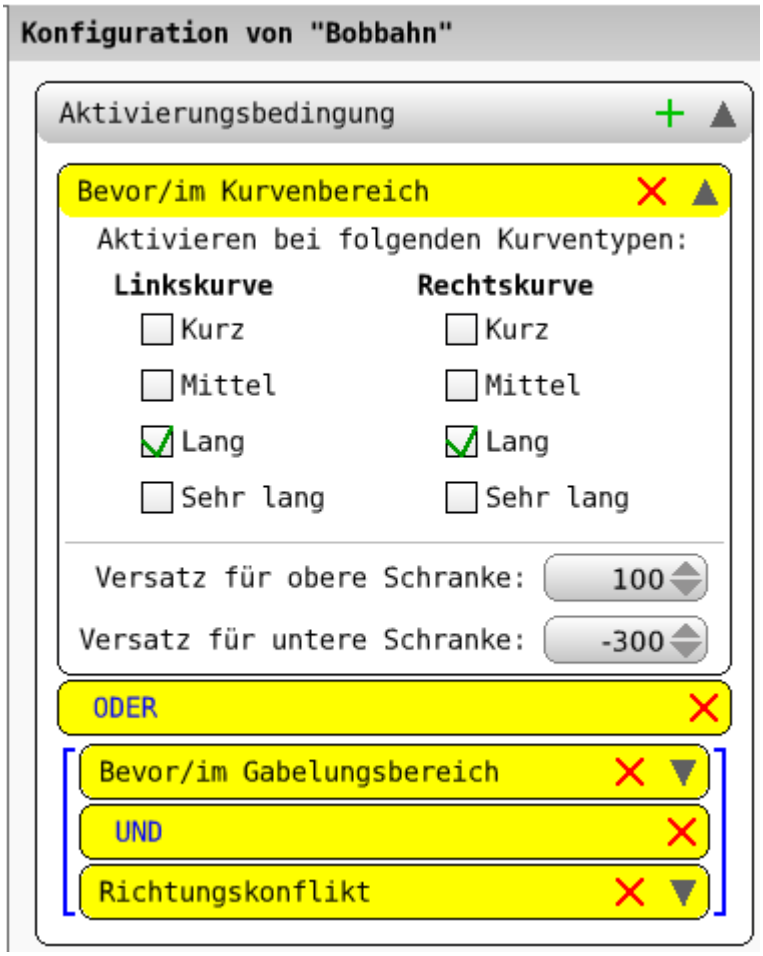


Abbildung 7.8: Erwartetes Ergebnis nach der Bearbeitung von Teilaufgabe 2 der Aufgabe 4.

Um einen gültigen Ausdruck herzustellen, müssen die Ereignisse verschoben werden.

7.4.5 *Interview*

Den Versuchspersonen wurden nach der Bearbeitung der Aufgaben folgende Fragen gestellt:

1. Wo gab es Probleme? Was war kontraintuitiv, schwer zu verstehen oder erschien umständlich?
2. Was war besonders einfach/positiv?
3. Wie wurde die Funktionalität/Gebrauchstauglichkeit generell wahrgenommen?
4. Waren die Lesbarkeit, das Layout und die Farben angemessen?

5. Unterstützt die Software Sie bei den Aufgaben, ohne Sie zu belasten?
6. Bietet die Software genug Erläuterungen von sich aus?
7. Wurden die Erwartungen und Gewohnheiten von bisher benutzter Software erfüllt?
8. War bei gemachten Fehlern der Korrekturaufwand gering?

Die letzten vier Fragen zielen auf die Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Erwartungskonformität und die Fehlertoleranz aus der Norm „DIN EN ISO 9241: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten; Teil 110 - Die Grundsätze der Dialoggestaltung“ ab (siehe Abschnitt 3.2.3). Es wurde nur nach vier von sieben Grundsätzen gefragt, um den zur Verfügung stehenden Zeitrahmen nicht zu sprengen und die Versuchspersonen auch nicht mit zu vielen Fragen zu belasten. [2]

7.4.6 *Ergebnisse*

Die Ergebnisse der Untersuchung wurden als eine Menge von Problemen festgehalten. Diese Probleme wurden, je nach Versuchsperson, entweder schon beim Studieren der Bedienungsanleitung und des Automatiken-GUIs, beim Bearbeiten der Aufgaben oder beim Interview am Ende festgestellt.

Es werden zunächst die positiven Bemerkungen zur Automatiken-GUI zusammengefasst:

- Drag and Drop zum Hinzufügen von Funktionen in den Ablaufplan wurde von allen vier Personen als positiv empfunden.
- Das dreiteilige Layout (Menüleiste an dieser Stelle ignoriert) des Automatiken-GUI war für alle vier Personen einfach verständlich.
- Durch die letzten vier Fragen konnten keine gravierenden Mängel an der Automatiken-GUI festgestellt werden.

Interessant ist auch, dass zum Beispiel der englische Dialog[3] zum Abspeichern einer Automatik keinem Benutzer negativ aufgefallen ist.

2 Der Grundsatz der Individualisierbarkeit ist ohnehin irrelevant, da das Automatiken-GUI keine Einstellungsmöglichkeiten bietet, die dem Grundsatz gerecht werden würden. Die Steuerbarkeit wird als gegeben angesehen. Die Selbstbeschreibungsfähigkeit wurde der Lernförderlichkeit vorgezogen.

3 Der erwähnte Dialog wurde mit einem Standard-Dialog-Widget von Squeak implementiert und konnte nicht lokalisiert werden.

Die festgestellten Probleme wurden den folgenden drei Kategorien zugeordnet und zogen entsprechende Verbesserungen in diesen Kategorien nach sich[4]:

1. *Probleme in der Bedienungsanleitung*
2. *Allgemeine Probleme im Automatiken-GUI*
   Alle Probleme im Automatiken-GUI, die nichts mit der Ereignis-Konfiguration zu tun haben.
3. *Probleme mit der Ereignis-Konfiguration*

Es folgt eine Auflistung der festgestellten Probleme. In Klammern ist jeweils angegeben, welche Versuchspersonen („Vpn“) die Probleme festgestellt haben. Die am häufigsten festgestellten Probleme werden innerhalb der Kategorien zuerst aufgelistet.

7.4.6.1 *Probleme in der Bedienungsanleitung*

PROBLEM.B1 (Vpn 2, 3, 4) *Abschnitt 4.7: Der Begriff „Boolescher Ausdruck“ ist unklar.*
GELÖST. „Boolescher Ausdruck“ wurde durch „UND/ODER-Ausdruck“ ersetzt. Da der Negationsoperator bisher nicht gebraucht wurde, ist „UND/ODER-Ausdruck“ hinreichend genau.

PROBLEM.B2 (Vpn 2, 3) *Abschnitt 4.7.1: Die beiden Einträge „(ODER)“ sowie „(UND)“ sind unklar.*
GELÖST. „(ODER)“ wurde zu „(ODER) [Verschachtelung]“ umbenannt. Analog für „(UND)“. Der Abschnitt in der Bedienungsanleitung sollte nun klarer sein, da als Beispiel auf ein Bild referenziert wurde, welches beide Operator-Typen enthält.

PROBLEM.B3 (Vpn 1, 4) *Abschnitt 4.8.: „Wo sind die fettgedruckten Bezeichner?“*
GELÖST. Nach der Erwähnung der „fettgedruckten Bezeichner“ wurde eine Referenz auf ein Bild eingefügt, welches die kombinierten Funktionen mit fettgedruckten Bezeichnern zeigt.

PROBLEM.B4 (Vpn 2, 3) *Abschnitt 9 -„Allgemeine Eigenschaften einer Automatik“ sollte weiter oben in der Anleitung stehen.*
GELÖST. Die zwei Versuchspersonen haben die Überschriften der Abschnitte überflogen und angemerkt, dass sie die „allgemeinen Eigenschaften“ weiter oben erwartet hätten. Die Bezeichnung „Allgemeine Eigenschaften“ ist ungenau, da es sich dabei nur um Name, Beschreibung und Kategorien der Automatik handelt. Entsprechend wurde der

4 Das Sammeln der Probleme und die Umsetzung der Verbesserungen wurden erst durchgeführt, nach dem alle vier Untersuchungen durchgeführt wurden.

Abschnitt in „Name, Beschreibung und Kategorien einer Automatik festlegen“ umbenannt.

PROBLEM.B5 (Vp 4) *Abschnitt 2: Die Bereiche des Automatiken-GUI sind im Bild schwer zu erkennen.*
GELÖST. Das Bild wurde vergrößert und die einzelnen Bereiche wurden in dem Bild durch rote Rechtecke hervorgehoben. In den Rechtecken befindet sich nun in roter Schrift der Name des Bereichs.

PROBLEM.B6 (Vp 4) *Abschnitt 4.2/4.4: Abschnitt „Verbindung entfernen“ sollte gleich nach „Verbinden von Funktionen“ kommen.*
GELÖST. Der Abschnitt „Verbindungen entfernen“ kommt nun direkt nach dem Abschnitt „Verbinden von Funktionen“. Analog für Abschnitt „Funktionen hinzufügen“ und „Funktionen entfernen“.

PROBLEM.B7 (Vp 2) *Abschnitt 4.2 und andere: Bilder enthalten die Funktion „Mauseingriffe“, welche aber nicht im Funktionsbereich gefunden werden kann.*
GELÖST. Die Funktion „Mauseingriffe“ wurde für die Untersuchung nicht freigeschaltet. Die Bilder wurden trotzdem ersetzt, da es sich dabei um eine Entwickler-Funktion handelt die von späteren Benutzern vermutlich nicht verwendet wird oder gar für diese nicht freigeschaltet wird. Die neuen Bilder zeigen die Funktion „Streckenvorschau“.

PROBLEM.B8 (Vp 2) *Abschnitt 4.3: „Warum sollen die Automatiken verbunden werden?“*
GELÖST. Folgender Satz wurde zur Klärung hinzugefügt: Um Funktionen in den Verarbeitungsablauf zu integrieren, müssen diese direkt oder indirekt mit “Anfang” und “Ende” verbunden sein.

PROBLEM.B9 (Vp 4) *Abschnitt 4.8: „Achtung“ irritiert.*
GELÖST. Das „Achtung“ an der Stelle warnt vor keiner Gefahr und ist daher irreführend. Es wurde entfernt.

PROBLEM.B10 (Vp 4) *Abschnitt 8 enthält eine ungültige Bild-Referenz.*
GELÖST. Die Referenz wurde entfernt, da sie sich auf ein nicht mehr vorhandenes Bild bezieht.

PROBLEM.B11 (Vp 4) *Erklärung und Bild sollten immer auf einer Seite erscheinen.*
TEILWEISE GELÖST. Da die Bedienungsanleitung relativ viele Bilder enthält, ist es schwierig diese direkt an dem Text zu positionieren, der sie referenziert, ohne große, leere Bereiche zu erzeugen. Einige Bildern konnten jedoch besser positioniert werden. An den Stellen, an denen das nicht möglich war, wurden die Referenzen angepasst, z. B. von „Abbildung X“ zu „Abbildung X auf Seite Y“.

7.4.6.2 *Allgemeine Probleme im Automatiken-GUI*

PROBLEM.A1 (Vpn 1, 2, 3, 4) *Dropdown-Menü: Menü öffnet sich nicht, wenn auf den Pfeil geklickt wird.*
GELÖST. Das Widget für das Dropdown-Menü funktioniert nun wie gewünscht.

PROBLEM.A2 (Vpn 2, 3, 4) *„Bobbahn" wurde nicht mit „von der Fahrbahn abkommen verhindern" assoziiert.*
GELÖST. Das Problem wurde Andreas Wickert, dem Implementierer dieser Funktion, geschildert und dieser hat bereits einen besseren Namen gefunden.

PROBLEM.A3 (Vpn 2, 4) *Konfigurationsbereich: Initial wird kein Scrollbalken angezeigt.*
GELÖST. Der Scrollbalken im Konfigurationsbereich wird nun ordnungsgemäß angezeigt, wenn eine Funktion im Ablaufplan ausgewählt wird.

PROBLEM.A4 (Vpn 1, 3) *Verbinden von Funktionen durch Ziehen und Loslassen auf den Verbindungspunkten.*
GELÖST. Diese Funktionalität wurde implementiert und steht als Alternative zur Benutzung zur Verfügung. In der Bedienungsanleitung wurde diese Alternative jedoch nicht dokumentiert, um die Bedienungsanleitung nicht unnötig zu verlängern.

PROBLEM.A5 (Vpn 2, 4) *Die Schrift in den Tooltips ist zu klein.*
ZU AUFWENDIG. Es konnte keine generische Möglichkeit gefunden werden die Schriftgröße zu ändern. Wegen Problem A.5 bietet es sich für die Tooltips an, eine eigene Implementierung bereitzustellen. Das wurde in diesem Rahmen als zu aufwendig erachtet.

PROBLEM.A6 (Vpn 2, 4) *Die runden Tooltips sind unschön.*
ZU AUFWENDIG. Siehe auch Problem A4.

PROBLEM.A7 (Vp 3) *Ein Pixelgenaues ablegen im Ablaufplan ist nicht möglich.*
ZU AUFWENDIG. Das Problem konnte nicht in einer vertretbaren Zeit gelöst werden.

PROBLEM.A8 (Vp 3) *Bei so vielen Funktionen wären Symbole hilfreich.*
ZU AUFWENDIG. Die Implementierung wurde in diesem Rahmen als zu aufwendig erachtet.

PROBLEM.A9 (Vp 1) *Dateinamen mit Leerzeichen können nicht abgespeichert werden.*
NICHT REPRODUZIERBAR. Sowohl auf GNU/Linux als auch auf Microsoft Windows XP konnte eine Datei mit

einem Namen bestehend aus mehreren Leerzeichen erfolgreich abgespeichert werden. Die gespeicherte Datei wurde danach im Automatiken-GUI auch angezeigt. Es ist daher zu vermuten, dass das eigentliche Problem ein anderes ist. Die Ursache konnte nicht identifiziert werden.

PROBLEM.A10 (Vp 3) *Das Löschen-Menü einer Verbindung soll auch dann angezeigt werden, wenn auf den Verbindungspunkt geklickt wurde.*
LÖSUNG PROBLEMATISCH. Das wäre möglich bei normalen Funktionen, aber die Funktion „Anfang" kann mehrere Ausgänge haben. Welcher sollte dann gelöscht werden? Analog für die Funktion „Ende". Diese Funktionalität nur für normale Funktionen umzusetzen würde eine Inkonsistenz einführen.

PROBLEM.A11 (Vp 4) *Die Schrift ist unmodern.*
ZU AUFWENDIG. Die Automatiken-GUI könnte in einem Einstellungsdialog diesen individuellen Vorzug berücksichtigen. Für diesen Rahmen wurde das aber als zu aufwendig eingestuft.

PROBLEM.A12 (Vp 3) *Eine Verbindung kann nicht durch Wegziehen dieser entfernt werden.*
GELÖST. Die Funktionalität ist nun implementiert und wurde in der Bedienungsanleitung anstatt der bisherigen Methode (Kontextmenü) dokumentiert, da dies intuitiver erscheint als das Kontextmenü zu benutzen.

7.4.6.3 *Probleme mit der Ereignis-Konfiguration*

PROBLEM.E1 (Vpn 1, 2) *Die Ereignisse und Operatoren können aus der Ereignis-Konfiguration herausgezogen werden.*
GELÖST. Wenn nun versucht wird die Ereignisse und Operatoren herauszuziehen, gleiten sie zurück an ihre ursprüngliche Position.

PROBLEM.E2 (Vpn 1, 2) *Beim Hinzufügen und Entfernen von Ereignissen oder Operatoren entstehen Darstellungsfehler/-artefakte.*
KEINE LÖSUNG. Dieses Problem ist bekannt und konnte nicht durch einen vertretbaren Aufwand gelöst werden. Auch durch Nachfrage in der Squeak-Community konnte keine Lösung gefunden werden.

PROBLEM.E3 (Vpn 1, 2) *Das Ereignis „Immer aktiv" ist nicht wiederherstellbar.*
GELÖST. Das Ereignis ist nun wiederherstellbar. Ein entsprechender Menüeintrag wurde dem Menü hinzugefügt.

PROBLEM.E4 (Vp 1) *Die Einträge „(ODER)“ sowie „(UND)“ sind unklar.*
GELÖST. Die Einträge im Menü heißen nun „(ODER) [Verschachtelung]“ sowie „(UND) [Verschachtelung]“.

PROBLEM.E5 (Vp 4) *Die Gültigkeit eines Ausdrucks sollte auch angezeigt werden.*
GELÖST. Es wird nun stets entweder „Bedingung gültig“ oder „Bedingung ungültig“ angezeigt.

PROBLEM.E6 (Vp 3) *Wenn die Ereignis-Konfiguration zugeklappt ist und durch das grüne Plus-Symbol Ereignisse oder Operatoren hinzugefügt werden, sieht man diese nicht.*
GELÖST. Ist der Aktivierungsbereich zugeklappt und klickt der Benutzer nun auf das grüne Plus-Symbol, wird der Bereich automatisch ausgeklappt.

PROBLEM.E7 (Vp 1) *Das Löschen eines Ereignisses kann nicht rückgängig gemacht werden.*
ZU AUFWENDIG. Ein UNDO-System ist wünschenswert, sollte allerdings für das ganze Automatiken-GUI eingeführt werden. Das wurde in diesem Rahmen als zu aufwendig erachtet.

PROBLEM.E8 (Vp 3) *Die Ereignis-Konfiguration ist zu kompliziert.*
LÖSUNG PROBLEMATISCH. Die Versuchsperson hat sich gefragt wieso Ereignisse konfiguriert werden müssen und nicht einfach eine Liste mit entsprechender Funktionalität zur Verfügung gestellt wird. Das wäre möglich, würde aber zu einer sehr langen Liste führen und würde die Flexibilität, die durch die Ereignis-Konfiguration eingeführt wurde, zunichte machen.

PROBLEM.E9 (Vp 3) *Es fehlt eine Vorschau, die zeigt wie sich die Aktivierungsbedingung in einem Versuchsschritt auswirkt.*
ZU AUFWENDIG. Diese Funktionalität ist wünschenswert und könnte mit einer Anpassung am vorgestellten Testframework für Ereignisse (siehe Abschnitt 7.1) umgesetzt werden. Das wurde in diesem Rahmen als zu aufwendig erachtet.

8 ZUSAMMENFASSUNG UND AUSBLICK

8.1 ZUSAMMENFASSUNG

Diese Arbeit ist die erste in einer Reihe[1], welche sich mit der Implementierung von Automatiken und Funktionen aus den zusammengetragenen Konzepten von Kain beschäftigt. Ziel dieser Arbeit war es, alle Funktionen mit weichen Eingriffen umzusetzen, die sich nicht mit dem „Anzeigen der aktuellen und optimalen Objektposition, -richtung und -geschwindigkeit und d[em] Anzeigen der aktuellen und optimalen Joystickposition der MWB“[2] beschäftigen.

Die Analyse in Kapitel 4 hat aufgezeigt, dass es mit 65 Funktionen relativ viele Funktionen zu implementieren galt. Die Funktionen sind sich teilweise recht ähnlich, unterscheiden sich nur im Detail. Es wurde auch erkannt, dass die Funktionen nur in gewissen Situationen eintreten.

Darauf aufbauend wurden in Kapitel 5 die Anforderungen zur Implementierung der Funktionen festgelegt. Die in der Analyse gefundenen Situationen wurden durch ein Ereignis-Modell verallgemeinert, sodass die Aktivierung einer Funktion von der eigentlichen Funktion entkoppelt werden konnte. Den Ungenauigkeiten bei der Beschreibung der Funktionen wurde mit Konfigurationsoptionen begegnet, sodass eine Menge von zu implementierenden „Hinweisfunktionen“ spezifiziert werden konnte. Die Menge der Hinweisfunktionen ist klein, aber durch die Konfiguration der passenden Hinweisfunktion(en) sollte es möglich sein, jede der zu implementierenden Funktionen zu realisieren.

Während der praktischen Umsetzung, dessen Ergebnisse in Kapitel 6 dokumentiert sind, wurde festgestellt, dass der Aufwand für die Implementierung des Ereignis-Systems und der Hinweisfunktionen unterschätzt wurde und somit nicht alle zu implementierenden Funktionen realisiert werden konnten. Wie in Anhang A festgehalten, wurden von den 65 zu implementierenden Funktionen 44 implementiert (und fünf weitere „versteckte Funktionen“ entdeckt).

Vermutlich hätten mehr Funktionen implementiert werden können, wenn das Ereignis-System nicht implementiert worden wäre. Die Einführung des Ereignis-Systems erschien jedoch ein sehr sinnvoller Schritt zu sein. Die Entkopplung einer Funktion

1 Weitere Arbeiten die folgen werden sind Seid (2012), Weidner-Kim (2012) und Wickert (2012).

2 vgl. Seid (2012)

von ihrer Aktivierung bringt nämlich drei Vorteile mit sich. Der Benutzer des Automatiken-GUI kann einerseits besser verstehen, wann eine Funktion aktiv wird (Transparenz), andererseits kann er ohne Programmieraufwand Funktionen mit Ereignissen kombinieren. Außerdem ist mit dem Ereignis-System auch ein kleines „Framework zur Aktivierung“ (siehe auch Anhang B) entstanden, wovon die anderen Implementierer in Seid (2012), Weidner-Kim (2012) und Wickert (2012) Gebrauch machen können bzw. es schon teilweise tun.

Während der praktischen Umsetzung wurde darauf geachtet, dass die Funktionalität auch durch Tests validiert ist (siehe Anhang 7). Das Automatiken-GUI und insbesondere die Ereignis-Konfiguration wurde in diesem Rahmen auch einer Untersuchung unterzogen. Es hat sich herausgestellt, dass in der Bedienungsanleitung, dem Automatiken-GUI insgesamt und der Ereignis-Konfiguration im Besonderen keine gravierenden Mängel festgestellt werden konnten. Sehr viele der festgestellten Probleme in dem Automatiken-GUI, der Ereignis-Konfiguration und der Bedienungsanleitung des Automatiken-GUI konnten behoben werden. Die Hinweisfunktionen sind dahingehend als getestet anzusehen, als das durch ihre Konfiguration die implementierten Funktionen (siehe Anhang A) umgesetzt werden konnten.

8.2 PROBLEME, IDEEN UND AUSBLICK

In Anhang A können alle Funktionen entnommen werden, die aus zeitlichen Gründen nicht implementiert werden konnten. Dies gilt ebenso für die folgenden Punkte.

8.2.1 *Implementierung „scheint“ in der XML-Konfigurationsdatei durch*

Siehe Abschnitt 6.1.2. Die Namen der Ereignis-Klassen sowie die Instanzvariablen sind direkt in der Konfigurationsdatei einsehbar. Wird eine Klasse oder Instanzvariable umbenannt, dann kann die Konfigurationsdatei nicht mehr eingelesen werden. Das Problem kann durch die Einführung einer Indirektion gelöst werden: Für die Klassen und Instanzvariablen sind Strings zu bestimmen, die statt den Namen der Klassen und Instanzvariablen hineingeschrieben werden.

8.2.2 *Darstellungfehler Event-Konfiguration*

Siehe Problem.E2 in Abschnitt 7.4.6. Dieser Fehler könnte möglicherweise durch ein Update auf eine aktuelle Squeak-Version

behoben werden. Dazu müsste jedoch SAM und AAF vollständig auf die neue Version portiert werden.

8.2.3 *Testframework für Funktionen erweitern*

Das in Abschnitt B.3 und 7.1 vorgestellte Testframework für Ereignisse könnte verallgemeinert werden, sodass Funktionen ebenfalls testbar gemacht werden können. Wie für die Ereignisse, könnten auch für die Funktionen IST-Ausgaben mit SOLL-Ausgaben verglichen werden.

8.2.4 *Video-Anleitung für das Automatiken-GUI*

Wie sich in den Vorbereitungen zu der durchgeführten Untersuchung herausgestellt hat, dauert das Lesen und Verstehen der Bedienungsanleitung des Automatiken-GUI ungefähr eine halbe Stunde. Diese Einarbeitungszeit könnte bedeutend reduziert werden, wenn die wesentliche Funktionalität des Automatiken-GUI in einem Video dokumentiert wäre.

LITERATURVERZEICHNIS

[Balzert 2000] Balzert, Helmut: *Lehrbuch der Software-Technik*. 2. Auflage. Spektrum Akademischer Berlag GmbH, 2000

[Bothe u. a. 2009] Bothe, Prof. Dr. K. ; Hildebrandt, Michael ; Niestroj, Nicolas: *ATEO-SYSTEM Komponente: SAMS*. 2009 https://www2.informatik.hu-berlin.de/swt/lehre/PR_MTI_0910/restricted/literature/Verhaltenspezifikation. – abgerufen am 26. Juni 2011, 15:42 Uhr

[Cooper u. a. 2010] Cooper, Alan ; Reimann, Robert ; Cronin, David: *About Face - Interface und Interaction Design*. mitp, 2010

[Dahm 2006] Dahm, Markus: *Grundlagen der Mensch-Computer-Interaktion*. Pearson Studium, 2006

[Denert 1992] Denert, Ernst: *Software-Engineering*. 1. korrigierter Nachdruck. Springer-Verlag, 1992

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

[Goldberg u. Robson 1989] Goldberg, Adele ; Robson, David: *Smalltalk-80 - The Language*. Bd. 1. Addison Wesley, 1989

[Gray u. Mohamed 1990] Gray, Philip D. ; Mohamed, Ramzan: *Smalltalk-80 - A Practical Introduction*. Bd. 1. Pitman, 1990

[Hasselmann 2012] Hasselmann, Michael: *(Titel unbekannt)*. 2012. – Diplomarbeit in Vorbereitung

[Heinecke 2004] Heinecke, Andreas M.: *Mensch-Computer-Interaktion*. Fachbuchverlag Leipzig, 2004

[Johnson 2008] Johnson, Jeff: *GUI Bloopers 2.0 - Common User Interface Design Don'ts and Dos*. 1. Elesevier, 2008

[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)*. 2011. – Studienarbeit

[Leonhard 2012] Leonhard, Christian: *Fenster zum Prozess: Weiterentwicklung eines Operateursarbeitsplatzes im Projekt Arbeitsteilung Entwickler Operateur (ATEO)*, Humboldt-Universtität zu Berlin, Diplomarbeit, 2012. – Diplomarbeit in Vorbereitung

[Maloney 2000] Maloney, John: *An Introduction to Morphic: The Squeak User Interface Framework*. http://stephane.ducasse.free.fr/FreeBooks/CollectiveNBlueBook/morphic.final.pdf. Version: 2000. – abgerufen am 19. Februar 2012, 19:55 Uhr

[Maloney 2011] Maloney, John: *Tutorial: Fun with the Morphic Graphics System*. http://static.squeak.org/tutorials/morphic-tutorial-1.html. Version: 2011. – abgerufen am 10. Februar 2012, 19:52 Uhr

[Nielsen 2005] Nielsen, Jakob: *Ten Usability Heuristics*. http://www.useit.com/papers/heuristic/heuristic_list.html. Version: 2005. – abgerufen am 17. Februar 2012, 18:21 Uhr

[Schwarz 2009] Schwarz, Hermann: *Fenster zum Prozess: ein Operateursarbeitsplatz zur Überwachung und Kontrolle von kooperativem Tracking*, Humboldt-Universität zu Berlin, Diplomarbeit, 2009. – Diplomarbeit

[Seid 2012] Seid, Aydan: *Konzeption und Implementierung von Algorithmen für Automatiken zur Prozessführung durch visuelle Anzeigen*. 2012. – Diplomarbeit in Vorbereitung

[Shneiderman u. Plaisant 2005] Shneiderman, Ben ; Plaisant, Catherine: *Designing The User Interface*. Pearson Education, 2005

[Sommerville 2011] Sommerville, Ian: *Software Engineering*. Ninth edition. Pearson Education, 2011

[Stritzinger 1997] Stritzinger, Alois: *Komponentenbasierte Softwareentwicklung*. Bd. 1. Addison Wesley Longman Verlag GmbH, 1997

[Weidner-Kim 2012] Weidner-Kim, Helmut: *(Titel unbekannt)*. 2012. – Diplomarbeit in Vorbereitung

[Wickert 2012] Wickert, Andreas: *(Titel unbekannt)*. 2012. – Diplomarbeit in Vorbereitung

ANHANG

A

ÜBERSICHT DER FUNKTIONEN

Die nachfolgende Tabelle dient mehreren Zwecken. Sie dient dem Überblick, da alle Funktionen und ihre zugehörigen Automatiken aus den Konzeptbögen von Kain (2012) aufgelistet werden, die im Rahmen dieser Arbeit zur Bearbeitung vorgesehen wurden. Durch diese Auflistung findet auch eine Abgrenzung zu der Arbeit von Seid (2012) statt, welcher spezielle visuelle Hinweise implementiert (siehe Abschnitt 1.2 und 4.1). Schließlich dokumentiert die Tabelle, welche Funktionen aus den Konzeptbögen implementiert werden konnten.

Die erste Spalte enthält die Nummer des Konzepts („K“) bzw. Teams. Die zweite Spalte enthält Automatiknamen (**fett** gedruckt), Funktionsnamen (normal gedruckt) und Kommentare zur Implementierung (blau gedruckt) jeweils in einer eigenen Zeile. Die Funktionsnamen sind eingerückt, sodass sie leicht der Automatik zugeordnet werden können. Ein Kommentar zur Implementierung bezieht sich auf die Funktion in der Zeile darüber und gibt u. a. an, welche Hinweisfunktion zur Implementierung (siehe Abschnitt C.12) benutzt wurde.

Bei einigen Konzepten heißen die Automatiken genau so wie die Funktionen (z. B. bei Konzept 24 - „Hinderniserkennung“ ist sowohl Automatik- als auch Funktionsname). Einige Funktionen heißen z. B. „Eingreifen“, beinhalten aber auch die Beschreibung von Hinweisen (z. B. bei Konzept 25). Einige Teams beschreiben auditive und/oder visuelle Hinweise in der allgemeinen Automatikbeschreibung auf den Konzeptbögen, ohne die Hinweise als eigentliche Funktion innerhalb des Automatik aufzuführen. Da bei diesen Fällen keine Funktion referenziert werden kann, wurde in der Tabelle stattdessen „(aus Automatikbeschreibung)“ vermerkt.

Insgesamt sind 65 Funktionen aus den Konzeptbögen aufgelistet. Fünf weitere Funktionen wurden aus der Automatikbeschreibung abgeleitet. 44 Funktionen wurden implementiert.

Die Konfigurationsdateien für die zusammengestellten Funktionen befinden im Verzeichnis „config.agents“. In den Dateinamen sind die Konzept-Nummern und Funktionsnamen kodiert.

K	AUTOMATIKEN UND FUNKTIONEN
20	**Auswahlassistent für Wegstrecken**
	Assistenzautomatik Wegstrecke
	Gabelungshinweis (visuell). Sehr ungenau, schwarze Pfeile genommen.
	Vermeiden von Kollision mit dynamischen Hindernis
	Warnung "BREMSEN" + auditiv
	Hinweis (auditiv). Hinweis (visuell).
	Vermeiden von Kollision mit statischen Hindernis
	(aus Automatikbeschreibung)
	Hindernisslalom (visuell).
22	**Gabelungsassistenz**
	Variantenanzeige
	Gabelungshinweis (visuell).
	Spurhalteassistenz
	(aus Automatikbeschreibung)
	Geschwindigkeitsassistenz / Kollisionsvermeidung
	"BRAKE" cue
	"SPEED" cue
23	**Gefahrenwarnung & -vermeidung**
	Warnung
	Vermeidung
	Handlungsanweisungen
	Anzeige
	Gabelungshinweis (visuell). Richtung unklar, bessere genommen. Schwarze Pfeile verwendet.
24	**Hinderniserkennung**
	Hinderniserkennung
	Entscheidungshilfe
	Entscheidungshilfe
	Gabelungshinweis (visuell). Richtung unklar, bessere genommen.
25	**Position**
	Anzeige Best Way
	Gabelungshinweis (visuell). Einschränkung: Anzeige bis „Richtung eindeutig“ nicht implementiert, stattdessen wird der Hinweis bis auf halber Höhe der Gabelungskachel angezeigt.
	Anzeige statisches Hindernis

Hindernisslalom (visuell).

Eingreifen

Geschwindigkeit

Anzeige dynamisches Hindernis

Hinweis (visuell).

Eingreifen

---

26 **Gabelung**

(aus Automatikbeschreibung)

---

27 **Lebenserhaltung I**

Hindernis statisch

Hinweis (visuell).

Hindernis dynamisch

Hinweis (visuell).

Gabel

Gabelungshinweis (visuell). „Zeichen links/rechts“ sehr ungenau. Es wird ein schwarzer Pfeil auf dem schnelleren Zweig angezeigt.

---

28 **Anzeige**

Gabelungsempfehlung

Randberührung

---

29 **Anzeige**

Vorschlag Richtung Gabelung

Gabelungshinweis (visuell). Richtung unklar, bessere genommen.

Alarm dynamisches Hindernis

---

30 **Navigation**

Helfer

Gabelungshinweis (visuell). Richtung unklar, bessere genommen.

Warner

Hinweis (visuell).

---

31 **Gabelungsassistent**

Alarm

Gabelungshinweis (visuell). Gabelungshinweis (auditiv). „Aufgrund des Fahrverhaltens“ ist unklar. Besseren Zweig genommen.

Tempoassistent

Hinweis

Hinweis (visuell).

---

32 **Mediator**

	Audioanweisung
34	**Ausgabe der Führungsgrößen**
	Richtungsvorgabe
	Gabelungshinweis (visuell). Richtung unklar, bessere genommen.
	Warnsignal
	Hinweis (auditiv).
35	**Gabelungshinweis**
	Einblenden des Pfeils
	Gabelungshinweis (visuell). Text und Bilder sind widersprüchlich. Der Text ist korrekt.
36	**Gabelungsentscheidung**
	Signale an den Fahrer
	Gabelungshinweis (visuell). Gabelungshinweis (auditiv).
37	**Streckenvorschau**
	Mini-Map (verkleinerte Streckenkarte)
	Streckenvorschau (visuell).
	Kurvenart ankündigen (inklusive Gabelungen)
	Gabelungshinweis (visuell). 4x Hinweis (visuell) mit den entsprechenden Ereignissen. „Starke Kurve“ wurde einer langen und sehr langen Kurve zugeordnet. „Leichte Kurve“ wurde mittleren Kurve zugeordnet. Die kurzen Kurven lassen sich nicht sinnvoll ankündigen.
	Hindernisse ankündigen (statische und dynamische)
	3 x Hinweis (visuell) mit den entsprechenden Ereignissen.
	Steuerung vorschlagen
	Fahrlinie einblenden
	Hindernisslalom (visuell). Gabelungshinweis (visuell).
	"schneller" durchsagen
	Hinweis (auditiv).
	"langsamer" durchsagen
	Hinweis (auditiv).
	eines von beidem durchsagen
	Hinweis (auditiv). Einschränkung: Es wird nur „schneller“ durchgesagt, wenn das möglich ist.
38	**Steering Assistanz - Command Generator**

	Path Advisor
	Gabelungshinweis (visuell).
	Speed Advisor dynamic obstacle
	Speed Advisor fixed obstacle
39	**Notfall**
	(aus Automatikbeschreibung)
40	**Personen Ausfall**
	visuelle Rückmeldung
	Hinweis (visuell).
	Virtuelle Leitplanke
	Ideallinie
	Aktivierung
	Kollisionshinweis
	Gabelungshinweis
	Gabelungshinweis (visuell) mit Standard-Positionierung. Besserer Zweig ausgewählt.
41	**Optimizer**
	Informer
	Gabelungshinweis (visuell). Richtung unklar, bessere genommen.
42	**Hinweis bei Gabelung**
	Wegweiser bei Gabelung
	Gabelungshinweis (visuell). Gabelungshinweis (auditiv).
	Warnung bei potentiellen Kollision (5)
	Kollision Warnung aural Kollision Warnung visuell
43	**Abstandsautomatik**
	Hinweis Eingriffsfunktion
	Gabelungsautomatik
	(aus Automatikbeschreibung)
	Gabelungshinweis (auditiv).
	Links-Rechts-Hinweis-Funktion
	Warnanzeige für scharfe Kurven / Gabelung und Hindernisse
	Warnhinweis Kurve, Hindernis, Gabelung
	3 x Hinweis (visuell) mit den entsprechenden Ereignissen.
44	**Karte**

	Karte
	Streckenvorschau (visuell).
	Akkustik-Assistent [sic]
	Akkustik-Assistent [sic]
	2 x Hinweis (auditiv) mit den entsprechenden Ereignissen.
	Moderations-Assistent
	Moderations-Assistent
45	**Entscheidungsassistent**
	Anzeige (1)
	Gabelungshinweis (visuell). Pfeile bevorzugt, da einfacher zu implementieren.
46	**XY-Longi-Lati-Control**
	Richtungsvorgabe
	Gabelungshinweis (auditiv). Richtung unklar, bessere genommen.
	Geschwindigkeitsinformation
	Hinweis (visuell).
47	**Gabelungsassistent**
	Signalisierung
	Gefahrenwarnung
	Hint Speed
	2 x Hinweis (visuell) mit den entsprechenden Hinweisen.
48	**Streckenführung**
	Gabelungsführung
	Gabelungshinweis (visuell).
	Hindernisführung
	Hindernisslaom (visuell). „An-/Ausschalter-abhängig“ unklar.
49	**Navigationshilfe**
	Steuerungsdisplay
	Entscheidungsvorschlag
	Gabelung Entscheidung
	Gabelungshinweis (visuell).
	Entscheidungsvorschlag
	Dynamisches Hindernis Entscheidung

B

ENTWICKLUNG VON EREIGNISSEN

Die aktuelle Version dieser Anleitung befindet sich im GIT-Repository des ATEO-Projekts im Verzeichnis *doc*[1].

B.1 EINFÜHRUNG

Im ATEO-Projekt werden mit dem Automatiken-GUI Funktionen zur Unterstützung der Mikroweltbewohner zusammengestellt und konfiguriert. Eine Funktion kann auf bestimmte Situationen während eines SAM-Versuchsschritts eingeschränkt bzw. zu diesen aktiviert werden. Das wird über sogenannte Ereignisse realisiert.

Diese Anleitung beschreibt, wie Ereignisse zu implementieren sind. In Abschnitt B.1.2 wird erklärt, welche Schritte dazu im Einzelnen notwendig sind. Zuvor werden die Ereignisse aus Sicht der Benutzer des Automatiken-GUI beschrieben.

B.1.1 *Allgemeines zu Ereignissen*

Ein Ereignis ist eine erkannte Situation während eines SAM-Versuchsschritts. Das Ereignis tritt ein, wenn die Situation erkannt wird, anderenfalls tritt das Ereignis nicht ein. Es wird zwischen Ereignissen mit Bezug zur Strecke und Ereignissen mit Bezug zum Fahrverhalten unterschieden. Erstere treten ein, sobald sich auf der Strecke bestimmte Streckenelemente (Kurven, Gabelungen oder Hindernisse) nähern oder das Objekt diese passiert. Ereignisse mit Bezug zum Fahrverhalten treten ein, sobald die Mikroweltbewohner durch ihre Joystick-Eingaben eine bestimmte Situation verursachen.

Ein Ereignis kann als boolesche Variable betrachtet werden. Ereignisse können verknüpft werden, sodass komplexe boolesche Ausdrücke entstehen können. Zur Verknüpfung stehen bisher die Operatoren *UND* sowie *ODER* aus der Aussagenlogik zur Verfügung. Der boolesche Ausdruck, der aus Ereignissen sowie den genannten Operatoren besteht und verwendet wird, um eine Funktion für einen Simulationsschritt zu aktivieren, heißt *Aktivierungsbedingung*.

Die Ereignisse werden in dem Automatiken-GUI vom Benutzer für eine Funktion konfiguriert und gegebenenfalls verknüpft. Ist im Ablaufplan des Automatiken-GUIs eine Funktion angeklickt,

1 http://www.assembla.com/code/ATEO/git/nodes/doc (abgerufen am 06. November 2011, 22:54 Uhr)

kann auf der rechten Seite die Aktivierungsbedingung der angeklickten Funktion festgelegt werden. Klickt der Benutzer mit der linken Maustaste auf das grüne Plus-Symbol, öffnet sich ein Menü, durch das Ereignisse und Operatoren dem Ausdruck hinzugefügt werden können. Abbildung B.1 zeigt das Menü mit den Einträgen.

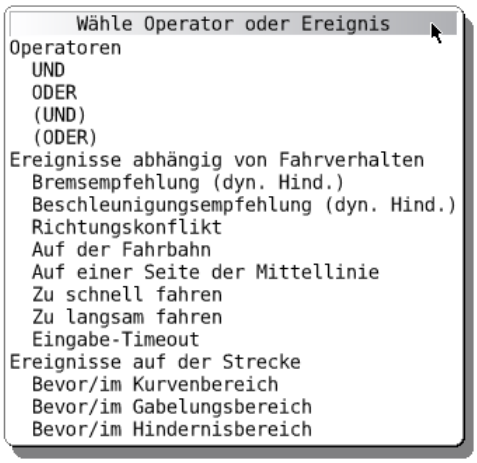


Abbildung B.1: Auswahlmenü zum Hinzufügen von Ereignissen und Operatoren für die Aktivierungsbedingung im Automatiken-GUI

Durch sinnvolle Anordnung der hinzugefügten Ereignisse und Operatoren, kann eine gültige Aktivierungsbedingung zusammengestellt werden. Abbildung B.2 zeigt ein Beispiel bestehend aus drei Ereignissen und den Operatoren *UND* und *ODER*. Wird mit einem Linksklick auf das dunkle Dreieck eines Ereignis-Widgets geklickt, erweitert sich das Ereignis-Widget um die einstellbaren Parameter des Ereignisses. Bei dem Ereignis *Eingabe-Timeout* kann der zu überwachende Mikroweltbewohner sowie das Timeout (in Millisekunden) konfiguriert werden.

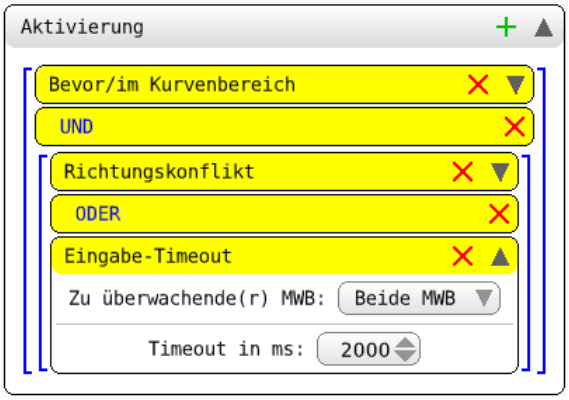


Abbildung B.2: Eine Aktivierungsbedingung mit den Operatoren UND sowie ODER im Automatiken-GUI

B.1.2 *Ziel dieser Anleitung*

Diese Anleitung beschreibt, wie im Kontext der AAF-Komponente des ATEO-Projekts ein neues Ereignis zu integrieren und zu testen ist. Im Einzelnen wird auf die folgenden drei Entwicklungsschritte eingegangen.

1. Implementierung eines Ereignisses (Abschnitt B.2),
2. Entwicklung von Tests für ein Ereignis (Abschnitt B.3)
3. Implementierung eines GUI zur Konfiguration eines Ereignisses (Abschnitt B.4)

Die Reihenfolge beschreibt das logisch sinnvolle Vorgehen bei der Entwicklung. Ein ungetestetes Ereignis sollte nicht in dem Automatiken-GUI erscheinen.

Es sei angemerkt, dass im Kontext der AAF-Komponente eine Funktion als *Agent* bezeichnet wird. Diese Bezeichnung wird auch in den folgenden Unterabschnitten verwendet.

B.2 ERSTELLEN EINES EREIGNISSES

Bei der Erstellung muss zwischen einem Ereignis mit Bezug zur Strecke und einem Ereignis mit Bezug zum Fahrverhalten unterschieden werden. Soll ein Ereignis mit Bezug zum Fahrverhalten erstellt werden, ist jeweils der Unterabschnitt *Ereignis mit Bezug zum Streckenelement* nicht relevant und umgekehrt.

B.2.1 *Klasse anlegen*

Ein Ereignis wird durch eine Ereignisklasse implementiert. Diese wird in der Kategorie *AAF-Events* angelegt und muss von der abstrakten Klasse `AAFAbstractEvent` oder `AAFAbstractTrackEvent` erben.

Ereignis mit Bezug zum Fahrverhalten

Der Name der Klasse sollte mit *AAFDriving* beginnen. Die Klasse ist von `AAFAbstractEvent` abzuleiten.

Ereignis mit Bezug zum Streckenelement

Der Name der Klasse sollte mit *AAFTrack* beginnen. Die Klasse ist von `AAFAbstractTrackEvent` abzuleiten.

B.2.2 *Initialisierung und Parameter*

Die Parameter werden als Instanzvariablen der Klasse realisiert. Um diese persistent in der Konfigurationsdatei eines Agenten

zu sichern, muss für jede Instanzvariable der Name (String) in der Methode `initialize` über `self persistentVariable:` bekannt gemacht werden (pro Instanzvariable ein Aufruf):

Listing B.1: AAFDrivingOnTrack initialize

```
AAFDrivingOnTrack initialize
  super initialize.
  minSensorsOnTrack := self class constantSensorCount.

  self persistentVariable: 'minSensorsOnTrack'.
```

Die Methode `initialize` soll dem Methodenprotokoll *initialize* zugeordnet werden.

Damit eine GUI-Klasse auf die Instanzvariablen der Parameter zugreifen kann, müssen diese über Getter- und Setter-Methoden verfügen. Diese Methoden sollten dem Methodenprotokoll *config* zugeordnet werden.

Ereignis mit Bezug zum Streckenelement

Die bisherigen Ereignisse mit Bezug zu Streckenelementen sind durch Benutzung des SAM-Distance-Dictionaries[2] implementiert. Für alle Ereignisse, die ebenfalls auf dem SAM-Distance-Dictionary basieren, müssen die relevanten Symbole dem Container `typesOfInterest` hinzugefügt werden:

Listing B.2: AAFTrackForkAhead initialize

```
AAFTrackForkAhead initialize
  super initialize.

  typesOfInterest addAll: { #nextElrFork . #nextRllFork }
```

B.2.3 *Eintreten eines Ereignisses*

Die Methode `isValid: aSamState` eines Ereignisses wird für jeden Simulationsschritt aufgerufen, um festzustellen, ob das Ereignis für den übergebenen SamState eintritt oder nicht. Tritt das Ereignis ein, soll der Rückgabewert der Methode `true`, anderenfalls `false`, sein. Die Methode ist dem Methodenprotokoll *implemented* hinzuzufügen.

Ereignis mit Bezug zum Fahrverhalten

Es ist die Methode `isValid: aSamState` zu implementieren. Für das Ereignis *Auf der Fahrbahn* sieht die Implementierung wie folgt aus.

2 Dieses Objekt entspricht der SAMModelData-Variable distanceDictionary.

Listing B.3: AAFDrivingOnTrack isValid:

```
AAFDrivingOnTrack isValid: aSamState

  | countSensorsOnTrack onTrack |
  countSensorsOnTrack := (self class constantSensorCount)
    - (aSamState noSensorsOffTrack).
  onTrack := countSensorsOnTrack >= minSensorsOnTrack.

  (onTrack) ifFalse: [
    self appendToLog: 'offTrack - ',
      minSensorsOnTrack asString, ' / ',
      countSensorsOnTrack asString.
  ].

  ^ onTrack.
```

Für Test- und Debug-Zwecke wird ein String geloggt, falls das Ereignis eintritt (mehr dazu in Abschnitt B.3.2).

Ereignis mit Bezug zum Streckenelement

Die Basisklasse `AAFAbstractTrackEvent` stellt bereits eine sinnvolle Implementierung von `isValid: aSamState` zur Verfügung. Für Ereignisse, die auf dem SAM-Distance-Dictionary basieren, sind keine weiteren Schritte notwendig. Um aber Testausgaben zu erzeugen (siehe Abschnitt B.3.2), sollte die Methode dennoch implementiert werden und die Basisklassenversion aufrufen:

Listing B.4: AAFTrackForkAhead isValid:

```
AAFTrackForkAhead isValid: aSamState

  | activated |
  activated := super isValid: aSamState.
  activated ifTrue: [
    self appendToLog: 'fork ', forkType asString.
  ].

  ^ activated
```

Für Ereignisse mit Bezug zu Streckenelementen, die nicht auf dem SAM-Distance-Dictionary basieren, muss die Methode `getDistanceObject: aSamState` implementiert werden. Als Rückgabewert wird ein Objekt vom Typ `SAMControllerDistanceObject` (Siehe Klassendokumentation) erwartet. Dieses Objekt kapselt alle Informationen die notwendig sind, damit die Basisklassenversion von `isValid: aSamState` korrekt funktioniert.

B.3 ERSTELLEN VON TESTS

B.3.1 *Überblick zum Testframework*

Um die Ereignisklassen sinnvoll zu testen, wurde ein Testframework entwickelt. Es basiert auf SUnit, dem bekannten Unit-Test-Framework von Smalltalk. Es kann sowohl für manuelle Tests als auch automatisiert ablaufende Tests verwendet werden. Für die automatisiert ablaufenden Tests werden einmalig die Joystick-Eingaben aufgezeichnet, um sie dann für den Test abspielen zu können.

Ein Test beruht auf dem Vergleich von Soll- und Ist-Daten. Die Ist-Daten der Ereignisse sind die Ausgaben, die von ihnen während eines SAM-Versuchsschritts erzeugt wurden. In Abschnitt B.3.2 wird beschrieben, wie Ereignisse mit Ausgaben versehen werden können.

Für den Test eines Ereignisses wird also ein SAM-Versuchsschritt gestartet. Liegen aufgezeichnete Joystick-Eingaben vor, werden sie vom Testframework für jeden Simulationsschritt in SAMModelData geschrieben, um für SAM echte Joystick-Eingaben zu simulieren.

Das Starten eines SAM-Versuchsschritts für einen Test erscheint unnötig. Es bringt aber folgende Vorteile mit sich:

1. *Vereinfachung von manuellen Tests*
   Das Testframework wurde zum Testen von Ereignissen entworfen. Es kann aber aus folgenden Gründen auch für manuelle Tests, auch von Agenten, verwendet werden.

   a) Ein SAM-Versuchsschritt kann gestartet werden, ohne dass sich der Tester durch SAM-Konfigurationsdialoge durchklicken muss. Auch der Countdown zu Beginn eines Versuchsschritts kann übersprungen werden.

   b) Das Objekt kann zu Beginn eines Versuchsschritts vor einem für den Test relevanten Streckenabschnitt positioniert werden. Das erspart dem Tester die Zeit, die das Objekt braucht, um bis zu dem gewünschten Abschnitt vorzudringen.

   c) Die Ablaufgeschwindigkeit für bestimmte Streckenabschnitte kann verändert werden. Hierdurch können irrelevante Abschnitte schneller übersprungen werden.

2. *Automatisierte Ereignistests können besser nachvollzogen werden*
   Zusätzlich zu dem, was SAM anzeigt, werden auf der rechten Seite weitere Informationen dargestellt. Dazu gehört ein vertikaler Balken, der anzeigt, ob das Ereignis gerade eintritt oder nicht und die visualisierten Joystick-Eingaben sowie die Ausgaben der Ereignisse (siehe Abschnitt B.3.4).

Durch diese zusätzlichen Informationen kann ein Test besser nachvollzogen werden, als wenn nichts angezeigt wird.

3. *Abdeckung von SAM- und AAF-Funktionalität*
   Durch die Tests wird SAM häufiger benutzt und somit können sich einschleichende Fehler schneller gefunden werden. Das Testframework wird durch einen speziellen Agenten realisiert. Bevor ein Test ablaufen kann, werden alle Einstellungen dieses Agenten und somit auch alle Einstellungen zum Test, in seine Konfigurationsdatei geschrieben. Durch die Nutzung des Testframeworks wird somit auch automatisch getestet, ob alle Parameter der Ereignisse ordnungsgemäß geschrieben und gelesen werden können.

Diesem Vorteil steht der Nachteil gegenüber, dass ein automatisiert ablaufender Test wegen des benötigten Versuchsschritts etwas mehr Zeit beansprucht. Dieser Nachteil kann jedoch durch die in Punkt 1 aufgeführten Maßnahmen verringert werden. Die meiste Zeit während eines SAM-Starts wird für das Laden der großen Streckenbilder benötigt. SAM wurde für das Testframework dahingehend optimiert, dass ein zuvor geladenes Streckenbild nicht noch einmal geladen wird, wenn es beim nächsten Versuchsschritt wieder benötigt wird. Wird für die Tests nur eine Strecke verwendet, dann wird diese auch nur einmal geladen. Diese Empfehlung wird auch in Abschnitt B.3.3 ausgesprochen.

B.3.2 *Ausgaben der Ereignisse*

Innerhalb einer Ereignisklasse soll die Methode `appendToLog: aString` benutzt werden, um eine Ausgabe für den aktuellen Simulationsschritt zu sichern. Der jeweils übergebene String wird den bisherigen Ausgaben als eine neue Zeile angefügt. Dabei wird dem übergebenen String in `appendToLog:` die `totalStepDistance` des SamStates vorangestellt. So wird gewährleistet, dass die Ausgabe mit einem Ortsbezug[3] (Streckenhöhe) gespeichert wird.

Damit die Ausgabe eines Ereignisses spezifisch genug ist, sollte sie unterscheidbar von den Ausgaben des gleichen Ereignisses anderer Konfigurationen sein. Es bietet sich an, einen Teil der Konfiguration als Information auszugeben. In `AAFTrackForkAhead` wird genau das erreicht, indem der Gabelungstyp übergeben wird:

Listing B.5: AAFTrackForkAhead isValid:

```
AAFTrackForkAhead isValid: aSamState

  | activated |
```

3 Implizit ergibt sich durch aufgezeichnete Joystick-Eingaben und die hier gespeicherte `totalStepDistance` auch ein Zeitbezug.

```
activated := super isValid: aSamState.
activated ifTrue: [
  self appendToLog: 'fork ', forkType asString.
].

^ activated
```

Es wird empfohlen nur eine Ausgabe zu generieren, wenn das Ereignis eintritt. In diesem Fall war es notwendig in der Ereignisklasse `AAFTrackForkAhead` die Methode `self isValid: aSamState` zu überschreiben (die schon eine sinnvolle Implementierung bereitstellt, jedoch ohne Ausgabe).

B.3.3 *Test erstellen*

Es wird beschrieben, welche Schritte notwendig sind, um einen Test für eine Ereignisklasse zu erstellen. Falls das Framework nur für manuelle Tests verwendet wird, ist die Durchführung des letzten Schritts Joystick-Eingaben aufzeichnen und Referenz-Ausgabe erzeugen nicht erforderlich.

B.3.3.1 *Testklasse anlegen*

Tests werden durch eine Testklasse implementiert. Der Name der Testklasse setzt sich zusammen aus dem Namen der zu testenden Klasse (die Ereignisklasse) und dem Zusatz *Test*. Die Testklasse der Ereignisklasse `AAFDrivingOnTrack` heißt entsprechend `AAFDrivingOnTrackTest`. Die Testklasse wird in der Kategorie *AAF-Events* angelegt und muss von der abstrakten Klasse `AAFEventTestCase` ableiten.

B.3.3.2 *Testmethode erstellen*

Ein Test wird durch eine Testmethode implementiert. Es wird empfohlen eine bereits existierende und funktionierende Testmethode zu kopieren und diese für den zu entwickelnden Test anzupassen.

Der Name einer Testmethode muss mit *test* beginnen und sollte widerspiegeln, was getestet wird. Zum Beispiel verfügt die Testklasse `AAFTrackObstacleAheadTest` u. a. über die Testmethoden `testDynamicObstacle`, `testSlalom` und `testStaticObstacle25Left`. Wie aus dem Namen hervorgeht, decken diese Testmethoden die Erkennung der verschiedenen Hindernisse ab.

Die Testmethode `testDynamicObstacle` wird vorgestellt.

Listing B.6: AAFTrackObstacleAheadTest testDynamicObstacle

```
testDynamicObstacle
  "self debug: #testDynamicObstacle"
```

```
| event |
"CONFIGURE EVENT"
event := AAFTrackObstacleAhead new.
event lowerBoundOffset: -800 upperBoundOffset: 100.
event typesOfInterest: { #nextDynamicObstacle }
    asOrderedCollection.

"SETUP AGENT"
self setupAgents:
  (Array
    with: (AAFInputProviderAgent withJoystickInputs: (self
        class testInput))
    with: (AAFTestAgent withDelegate: event)
  ).

"SETUP STEP"
self configureStepMwiControl: '1'.
self configureStepTrack: 'hauptabschnitt2'.
self configureStepObstacleConfig: 'obstacleConfig_2'.
self configureStepStartAtDistance: 8500.
self configureStepEndAtDistance: 26100.
self configureStepTickDuration: #( #(0 99999999 5)).

"RUN STEP"
self runStep.

"PRINT & COMPARE OUTPUT"
self printAndCompareOutputWith: self class
    testOutputReferenceDynamicObstacle.
```

Der Quelltext der Testmethode folgt grob einer festen Struktur. Die verschiedenen Abschnitte werden durch großgeschriebene Kommentare eingeteilt:

- *CONFIGURE EVENT*
  Das zu testende Ereignis wird konfiguriert. Dazu wird eine Instanz der Ereignisklasse `AAFTrackObstacleAhead` erstellt und in den nachfolgenden Zeilen konfiguriert. Der Versatz für die untere Schranke wird auf 800 Pixel vor dem Hindernis und der Versatz für die obere Schranke wird auf 100 Pixel danach festgelegt.

- *SETUP AGENT*
  In diesem Abschnitt werden die Hilfsagenten konfiguriert, die den Test ermöglichen. Der erste Agent ist vom Typ `AAFInputProviderAgent` und simuliert aufgezeichnete Joystick-Eingaben. Die Joystick-Eingaben liegen als String vor und werden aus der Klassenmethode `testInput` bezogen. Ohne diesen Agenten wären automatisch ablaufende Tests nicht möglich.
  Der zweite Agent vom Typ `AAFTestAgent` nimmt das konfigurierte Ereignis entgegen und integriert es in den Versuchs-

schritt. Damit das Testframework ordnungsgemäß funktioniert, sollte dieser Agent immer mit einem gewünschten Ereignis übergeben werden.

- *SETUP STEP*
  Der SAM-Versuchsschritt wird konfiguriert. Es wird die Steuergewalt, die Strecke und die Hinderniskonfiguration festgelegt. Zusätzlich, um den Test schneller ablaufen zu lassen, wird die Position des Objekts zu Beginn des Versuchsschritts auf 8500 Pixel der Streckenhöhe festgelegt. Dadurch wird das Objekt kurz vor dem ersten dynamischen Hindernis auf der Strecke positioniert. Bei Streckenhöhe 26100px wird der Versuchsschritt beendet. Das Simulationsschrittintervall wird für die ganze Strecke auf 5ms[4] gesetzt. Der Hindernistyp wird auf `#nextDynamicObstacle` gesetzt, damit das Ereignis beim dynamischen Hindernis eintritt. Es wird empfohlen die Strecke 'hauptabschnitt2' für die Tests zu benutzen. Durch die Festlegung auf eine Strecke für alle Tests, muss nur eine Strecke für alle Tests geladen werden.

- *RUN STEP*
  Der Versuchsschritt wird gestartet und läuft mit dem konfigurierten Ereignis ab. Alternativ kann der Versuchsschritt mit `self runStepProfiled` gestartet werden, woraufhin das Laufzeitverhalten mit `MessageTally`[5] analysiert wird. Die Ausgabe von `MessageTally` wird in einer Datei abgespeichert.

- *PRINT & COMPARE OUTPUT*
  Die generierte Ist-Ausgabe des Ereignisses wird mit einer Soll-Ausgabe verglichen. Die Soll-Ausgabe steht über die Klassenmethode `testOutputReferenceDynamicObstacle` zur Verfügung.

Die einzelnen Methoden des Testframeworks werden in Abschnitt B.3.5 beschrieben.

B.3.3.3 *Joystick-Eingaben aufzeichnen und Referenz-Ausgabe erzeugen*

Diese beiden Schritte können zusammen erledigt werden. Die Testmethode wird soweit vorbereitet, dass das Ereignis konfiguriert ist und auch die anderen Einstellungen sinnvoll getätigt wurden. Da noch keine Joystick-Eingaben aufgezeichnet wurden

4 Das Simulationsschrittintervall ist normalerweise auf 39ms gesetzt. Das Setzen des Simulationsschrittintervall auf 5ms beschleunigt den Versuchsschritt erheblich.

5 Die Benutzung von MessageTally ist unter Squeak der gängige Weg um das Laufzeitverhalten zu analysieren.

und keine Referenz-Ausgabe vorliegt, kann `setupAgents:` kein `AAFInputProviderAgent`-Objekt übergeben werden und die Methode `printAndCompareOutputWith:` kann nicht aufgerufen werden. Entsprechende Stellen im Quelltext der bisherigen Testmethode sind daher auszukommentieren.

1. *Testmethode ausführen und Daten generieren*
   Die bisherige Testmethode wird ausgeführt (siehe B.3.4). Dadurch wird ein SAM-Versuchsschritt gestartet. Der Tester sollte mit geeigneten Joystick-Auslenkungen die Erkennung des Ereignisses testen. Dabei sollte auch darauf geachtet werden, dass die Ausgabe des Ereignisses den Erwartungen entspricht.
   Eventuell können durch diesen Lauf einige Einstellungen optimiert werden (z. B. für `configureStepTickDuration:` oder `configureStepStartAtDistance:`). In diesem Fall wird die Methode noch einmal ausgeführt.
   Da bei dieser Ausführung nichts verglichen wird, fällt der Test erfolgreich aus.

2. *Methode für Joystick-Eingaben erstellen*
   Durch den ausgeführten manuellen Test in Schritt 1 stehen die für den Test gewünschten Joystick-Eingaben in Logdateien von SAM zur Verfügung. Diese werden durch eine Methode abrufbar gemacht.
   Es wird eine Klassenmethode angelegt, z. B. `testInput`. In diese Methode wird die Zeile `super lastJoystickInputs.` geschrieben und markiert. Anschließend wird die Zeile mit der Tastenkombination Strg+P ausgegeben. Am Ende der Zeile erscheint der markierte String mit den Joystick-Eingabedaten. Die Markierung wird aufgehoben, der mit der Tastenkombination Strg+P ausgeführte Quelltext wird nun gelöscht oder auskommentiert und vor den String wird das Zeichen '^' eingefügt, um den gesamten String als Rückgabewert auszuweisen. Die fertige Methode sieht wie folgt aus (gekürzt):

Listing B.7: Joystick-Eingabedaten in AAFTrackObstacleAheadTest

```
AAFTrackObstacleAheadTest testInput

  "super lastJoystickInputs."

  ^
'527@0 0@0
664@0 0@0
664@-85 0@0
495@-85 0@0
```

```
442@-107 0@0
'
```

3. *Methode für Referenz-Ausgabe erstellen*
   Die Ausgabe des Ereignisses vom letzten Aufruf wurde zwischengespeichert und kann als Referenz-Ausgabe verwendet werden. Es wird eine Klassenmethode angelegt, z. B. `testOutputReference`. In diese Methode wird die Zeile `super lastOutput.` geschrieben. Es wird fortgefahren wie in Schritt 2. Die fertige Methode sieht wie folgt aus.

Listing B.8: Soll-Ausgabe in AAFTrackObstacleAheadTest

```
AAFTrackObstacleAheadTest testOutputReference

  ^
'9073: obstacle nextDynamicObstacle
9084: obstacle nextDynamicObstacle
9094: obstacle nextDynamicObstacle
9893: obstacle nextDynamicObstacle
'
```

Nun kann in der Testmethode `setupAgents:` ein mit `self class testInput` parametrisiertes `AAFInputProviderAgent`-Objekt übergeben werden. Damit die Ist- und Referenz-Ausgabe verglichen werden, muss am Ende der Methode `printAndCompareOutputWith:` mit der erzeugten Referenz-Ausgabe (Methode `self class testReferenceOutput`) aufgerufen werden.

Der Test kann nun noch einmal ausgeführt werden. Er sollte erfolgreich verlaufen. Zur Sicherheit sollte die Referenz-Ausgabe verändert werden (z. B. durch Hinzufügen eines Zeichens) und der Test noch einmal laufen gelassen werden, um sicherzugehen, dass der Test auch fehlschlagen kann. Funktioniert alles, sollte die Änderung wieder rückgängig gemacht werden.

Der Versuchsschritt wird vorzeitig beendet, wenn ein Agent vom Typ `AAFInputProviderAgent` der Methode `setupAgents:` übergeben wurde und für weitere Simulationsschritte keine Joystick-Eingaben vorliegen. Aus diesem Grund empfiehlt es sich die in `testInput` gespeicherten Joystick-Eingaben zu kürzen (iteratives Vorgehen). Dadurch läuft der Test nicht länger als nötig.

Somit ist ein automatisiert ablaufender Test erstellt worden.

B.3.4 *Tests ablaufen lassen*

Die Tests können über den Test Runner von SUnit ausgeführt werden. Da für den Test ein SAM-Versuchsschritt gestartet wird,

kann der Test mitverfolgt werden. Auf der rechten Seite des 800x768 Pixel großen Sichtbereichs von SAM werden zusätzliche Informationen eingeblendet (Abbildung B.3). Diese sind (von oben nach unten):

- Ereignisklasse und Testmethode
- Sektion *Configuration*
  - Streckenkonfiguration (Streckenname, Hinderniskonfiguration, Steuergewalt)
  - Starthöhe des Objekts auf der Strecke (Start Distance) sowie veränderte Simulationsschrittintervalle für bestimmte Streckenabschnitte (Tick Durations)
  - Ob Joystick-Eingaben abgespielt werden (Input Provider)
- Sektion *Joystick inputs*
  Für jeden Mikroweltbewohner, für den die Joystick-Eingaben von SAM verarbeitet werden, wird die aktuelle Joystick-Auslenkung angezeigt. Werden die Joystick-Eingaben von beiden Mikroweltbewohnern verarbeitet, so wird auch die Auslenkung für die Gesamtlenkung angezeigt.
- Sektion *Output*
  Die Ausgabe des getesteten Ereignisses. Aus Performancegründen werden nur die letzten drei Zeilen angezeigt[6].
- *Step Distance*
  Die zurückgelegte Distanz in Pixeln relativ zur unteren Kante des Sichtbereichs (entspricht `totalStepDistance` aus SAMModelData).

Zwischen dem, was SAM anzeigt und den beschriebenen zusätzlichen Informationen befindet sich zusätzlich ein vertikaler Balken. Er ist schwarz (wie in der Abbildung), wenn das Ereignis nicht eintritt, und hellgrün, wenn das Ereignis eintritt. Dieser Indikator erweist sich vor allem beim manuellen Testen von Ereignissen als sehr nützlich.

B.3.5 *Methoden des Testframeworks*

In diesem Abschnitt werden die Methoden des Testframeworks beschrieben, die in den Testmethoden aufgerufen werden können. Für jede Methode wird deren Zweck und ihre Parameter dokumentiert. Die Methoden haben keine sinnvollen Rückgabewerte.

6 Die Ausgabe von vielen Zeichenketten in kurzer Zeit führt zu einer merklichen Leistungsminderung Laufzeitumgebung.

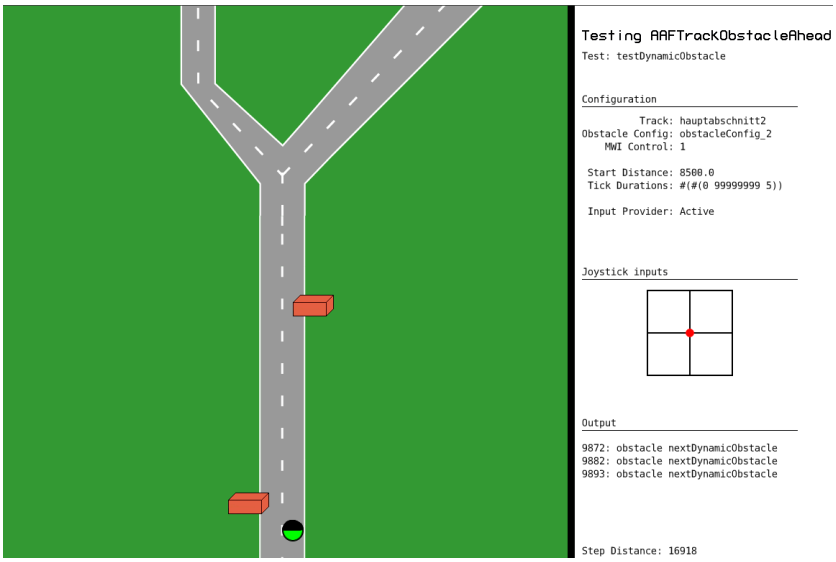


Abbildung B.3: Ablauf eines Tests. Auf der rechten Seite sind zusätzliche Informationen eingeblendet.

Der Aufruf der Methoden `configureStepStartAtDistance:` und `configureStepTickDuration:` ist optional. Sie dienen nur der Beschleunigung des Ablaufs des Versuchsschritts. Alle anderen Methoden sollten in der Testmethode aufgerufen werden, wobei von den beiden `runStep`-Methoden nur eine ausgeführt werden sollte. Die Methoden müssen in der Reihenfolge aufgerufen werden, wie sie nachfolgend beschrieben werden.

- **`configureStepMwiControl`**`: aMwiControl`
  Legt fest, ob die Joystick-Daten nur von Mikroweltbewohner 1, nur von Mikroweltbewohner 2 oder von beiden beachtet werden. Gültige Argumente sind: '1', '2' und '12' (Strings).

- **`configureStepTrack`**`: aTrack`
  Legt die Bilddatei für die Strecke fest. Diese wird als String ohne Punkt und Dateiendung übergeben. Ein gültiger Wert ist z. B. 'hauptabschnitt2'. Es wird empfohlen die Strecke 'hauptabschnitt2' für die Tests zu benutzen, da auf dieser nicht nur alle Gabelungstypen zum Einsatz kommen, sondern auch alle Hindernistypen. Durch die Festlegung auf eine Strecke für alle Tests, muss nur eine Strecke für alle Tests geladen werden[7].

- **`configureStepObstacleConfig`**`: anObstacleConfig`
  Legt die Hinderniskonfigurationsdatei fest. Diese wird als

7 Die meiste Zeit beim Laden eines Versuchsschritts wird benötigt, um die Bilddatei für die Strecke zu laden. SAM wurde für das Testframework dahingehend optimiert, dass ein zuvor geladenes Streckenbild nicht noch einmal geladen wird, wenn es beim nächsten Versuchsschritt wieder benötigt wird. Die Festlegung einer Strecke für alle Tests beschleunigt den Ablauf aller Tests daher erheblich.

String ohne Punkt und Dateiendung übergeben. Ein gültiger Wert ist z. B. 'obstacleConfig_0'.

- **`configureStepStartAtDistance`**`: distanceInPx`
  Legt fest, bei welcher Streckenhöhe der Versuchsschritt gestartet wird. Das Argument entspricht einem Wert der Instanzvariablen `totalStepDistance` aus SAMModelData.

- **`configureStepEndAtDistance`**`: distanceInPx`
  Legt fest, bei welcher Streckenhöhe der Versuchsschritt beendet wird. Das Argument entspricht einem Wert der Instanzvariablen `totalStepDistance` aus SAMModelData.

- **`configureStepTickDuration`**`: aTickDurationConfig`
  Legt fest, für welche Streckenabschnitte welches Simulationsschrittintervall verwendet wird. Als Argument wird ein Array von Arrays erwartet. Ein inneres Array besteht aus genau drei Elementen. Die ersten beiden Elemente entsprechen den Begrenzungen des Streckenabschnitts (Streckenhöhen). Das dritte Element ist das Simulationsschrittintervall in Millisekunden.
  Ein Beispiel ist #(#(0 1000 5) #(3000 9000 10)). Zwischen Streckenhöhe 0 und 1000 wird das Simulationsschrittintervall auf 5ms eingestellt. Zwischen Streckenhöhe 3000 und 9000 wird das Simulationsschrittintervall auf 10ms eingestellt.
  Die spezifizierten Streckenabschnitte dürfen sich nicht überschneiden.

- **`setupAgents`**`: agents`
  Ein Array von Agenten wird in der übergebenen Reihenfolge in den AAF-Graphen integriert. Der erste Agent wird mit der Quelle verbunden, der letzte mit der Senke. Die Agenten dazwischen (bei mehr als zwei Agenten) werden nacheinander miteinander verbunden.
  Zwei besondere Agenten sind Teil des Testframeworks:

  - *Agent vom Typ* `AAFInputProviderAgent`
    Dieser Agent gaukelt SAM Joystick-Eingaben vor. Mit der Klassenmethode `withJoystickInputs: inputAsString` können die Joystick-Eingaben als String gesetzt werden. Wird dieser Agent benutzt, haben manuelle Joystick-Eingaben keinen Einfluss, da diese von diesem Agenten überschrieben werden. Alternativ kann die Klassenmethode `withNormalJoystickInputs` verwendet werden, um Eingaben in Ruhestellung des Joysticks zu generieren. Dieser Agent sollte stets vor dem Agenten des Typs `AAFTestAgent` übergeben werden.

– *Agent vom Typ* `AAFTestAgent`
Dieser Agent ist zwingend erforderlich, damit ein Ereignis getestet werden kann. Er wird mit dem zu testenden Ereignis durch die Klassenmethode `withDelegate :aDelegateAgent` parametrisiert.
Ein typisches Argument für den Aufruf dieser Methode ist das Folgende:

```
(Array
  with: (AAFInputProviderAgent withJoystickInputs:
      (self class testInput))
  with: (AAFTestAgent withDelegate: event)).
```

Von den beiden Agenten des Frameworks sollte immer nur eine Instanz übergeben werden. Es können zusätzliche Agenten hinzugeschaltet werden, diese sind für den Test jedoch nicht relevant. Ist kein Joystick verfügbar, bietet es sich bei manuellen Tests an, einen Agenten vom Typ `AAFMouseInputAgent` zu integrieren. Dieser sollte vor dem Agenten des Typs `AAFTestAgent` geschaltet werden.

- **`runStep`**
Startet den konfigurierten Versuchsschritt.

- **`runStepProfiled`**
Alternative zu `runStep`. Wie `runStep`, aber der Versuchsschritt wird in `MessageTally spyOn: [ ... ]` gekapselt, sodass das Laufzeitverhalten analysiert wird. Die Ausgabe von `MessageTally` wird in eine Datei geschrieben und das Verzeichnis in dem die Datei abgelegt wird, ist das gleiche, in dem die Datei Squeak.exe liegt. Der Dateiname fängt mit *ProfiledStep* an und enthält den Namen der Testklasse sowie die Startzeit des Versuchsschritts. Ein Beispiel ist 'ProfiledStep_AAFTrackForkAheadTest_17:19:15.txt'.

- **`printAndCompareOutputWith`**`: referenceOutput`
Vergleicht die erzeugten Ist-Ausgaben eines Ereignisses mit den als String übergebenen Soll-Ausgaben. Das Ergebnis des Vergleichs wird über `Transcript` geschrieben und kann in einem zuvor geöffneten Transcript-Fenster eingesehen werden. Zu den ausgegeben Informationen gehören: Ereignisklasse, Testmethode, Start- und Endzeit des Tests, Ergebnis des Ist-Soll-Vergleichs, die Informationen aus der Sektion *Configuration* (siehe Abschnitt B.3.4). Ein Beispiel ist folgende Ausgabe:

Listing B.9: Beispiel für eine Log-Ausgabe

```
================================================
Test of AAFTrackForkAhead
------------------------------------------------
    test method: testBothForks
     started at: 5:37:44 pm
    finished at: 5:38:05 pm
    test result: PASS
------------------------------------------------
          Track: hauptabschnitt1
Obstacle Config: obstacleConfig_0
    MWI Control: 1

 Start Distance: 16625.0
 Tick Durations: #(#(0 99999999 5))

 Input Provider: Active
------------------------------------------------
16625: fork nextElrFork
16635: fork nextElrFork
17567: fork nextElrFork

================================================
```

B.4 ERSTELLEN EINES GUI

Damit ein Ereignis in dem Automatiken-GUI verwendet und konfiguriert werden kann, muss eine GUI-Ereignisklasse implementiert werden. Das Automatiken-GUI basiert auf dem Morphic-Framework von Squeak. Auf dem Morphic-Framework aufbauend, wurden für das Automatiken-GUI Widgets erstellt (Kategorie *AAFGT-Widgets*), die auch von den Ereignis-GUIs benutzt werden sollen. An dieser Stelle sei insbesondere auf die Klasse `AAFGTWidgetGallery` verwiesen. Sie demonstriert, welche Widgets existieren[8] und wie sie verwendet werden können.

Es sei angemerkt, dass das Automatiken-GUI von deutschsprachigen Benutzern verwendet wird. Alle in der GUI sichtbaren Strings sollten daher in Deutsch sein.

B.4.1 *Klasse anlegen*

Der Name der GUI-Ereignisklasse setzt sich zusammen aus dem Namen der Ereignisklasse und dem Zusatz *GUI*. Die GUI-Ereignisklasse der Ereignisklasse `AAFDrivingOnTrack` heißt entsprechend `AAFDrivingOnTrackGUI`. Die GUI-Ereignisklasse wird in der

8 Mit `AAFGTWidgetGallery open` werden alle verfügbaren Widgets angezeigt.

Kategorie *AAFGT-Events* angelegt und muss von der abstrakten Klasse `AAFAbstractEventGUI` erben.

B.4.2 *Name und Beschreibung des Ereignisses im GUI*

Der Name des Ereignisses in dem GUI wird über die Klassenmethode `plainTextName` zurückgegeben. Über die Klassenmethode `helpText` wird ein Hilfetext zurückgegeben. Dieser erscheint als Tooltip, sobald der Benutzer mit dem Mauszeiger einige Sekunden über dem Ereignis verweilt.

In `AAFDrivingOnTrackGUI` sind diese Klassenmethoden wie folgt implementiert:

Listing B.10: AAFDrivingOnTrack plainTextName

```
AAFDrivingOnTrackGUI plainTextName
  ^ 'Auf der Fahrbahn'

AAFDrivingOnTrackGUI helpText
  ^ 'Ereignis tritt ein, sobald N oder mehr Sensoren melden,
      dass das Objekt sich auf der Fahrbahn befindet.'
```

B.4.3 *Methode setupConfigWidget implementieren*

In der Methode `setupConfigWidget` gilt es, die Instanzvariable `configWidget` aus der Basisklasse `AAFDrivingOnTrackGUI` mit einem `Morph` (oder einer Subklasse) zu initialisieren, der die Konfiguration des Ereignisses ermöglicht. Dazu kann auf zahlreiche Widgets aus der Kategorie *AAFGT-Widgets* sowie die Klasse `AAFEventUtils` zurückgegriffen werden.

Die Methode `setupConfigWidget` von `AAFDrivingOnTrackGUI` ist wie folgt implementiert.

Listing B.11: AAFDrivingOnTrackGUI setupConfigWidget

```
AAFDrivingOnTrackGUI setupConfigWidget
  | controlSpinButton label |

  label := LabelStringMorph
    contents: 'Mindestanzahl der Sensoren: '.
  controlSpinButton := AAFGTSpinButton
    value: event minSensorsOnTrack
    range: (Array with: 1 with: 8)
    climbRate: 1
    target: event
    action: #minSensorsOnTrack:.

  configWidget := AAFEventUtils configWidgetRow.
  configWidget
    addMorphBack:
```

```
    (AAFGTWidgetUtils rightAligned: label);
  addMorphBack:
    (AAFGTWidgetUtils leftAligned: controlSpinButton).
```

Mit einem Spinbutton (`AAFGTSpinButton`) kann über die Getter- und Settermethoden `minSensorsOnTrack` und `minSensorsOnTrack:` der Parameter des Events konfiguriert werden. Vor dem Spinbutton wird ein Label angebracht und ausgerichtet. Das Konfigurationswidget für dieses Ereignis ist in Abbildung B.4 zu sehen.

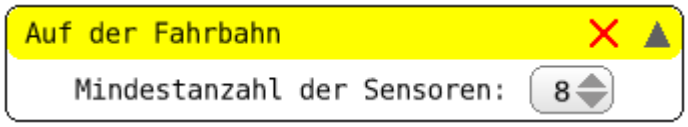


Abbildung B.4: Konfigurationswidget (Ereignis-GUI) für das Ereignis *Auf der Fahrbahn*

Damit ist die Entwicklung einer GUI zum Konfigurieren der Parameter eines Ereignisses abgeschlossen.

C

BEDIENUNGSANLEITUNG AUTOMATIKEN-GUI

Die aktuelle Version dieser Bedienungsanleitung befindet sich im GIT-Repository des ATEO-Projekts im Verzeichnis *doc*[1].

C.1 ALLGEMEINES

Diese Bedienungsanleitung ist für die Benutzungsschnittstelle *Konfigurationstool der Automatik-Funktionen (ATEO)*, im Folgenden *Automatiken-GUI* genannt, gedacht.

Mit der Automatiken-GUI können Automatiken konfiguriert werden, welche der Prozessüberwachung und -führung von SAM durch indirekte Eingriffe über Informationen an die Mikrowelt-bewohner oder direkte Eingriffe in die Steuerung des Objekts dienen und einen Operateur ersetzen sollen. Im Automatiken-GUI stehen eine Menge von Funktionen zur Verfügung, von denen flexibel Gebrauch gemacht werden kann. Diese können sinnvoll kombiniert werden, um eine gewünschte Automatik umzusetzen.

Neben Funktionen die Hinweise geben, gibt es zum Beispiel auch Funktionen die in die Steuerung korrigierend eingreifen.

C.2 STARTEN DES AUTOMATIKEN-GUI

Das Automatiken-GUI wird gestartet, indem in einem Workspace-Fenster in Squeak Folgendes ausgeführt wird:

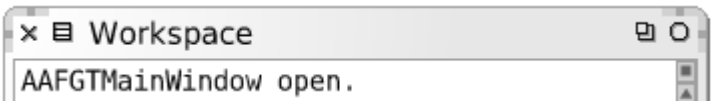


Abbildung C.1: Starten des Automatiken-GUI

1 http://www.assembla.com/code/ATEO/git/nodes/doc (abgerufen am 06. November 2011, 22:54 Uhr)

C.3 AUFBAU DES AUTOMATIKEN-GUI

Das Automatiken-GUI ist in folgende Bereiche unterteilt (siehe Abbildung C.2):

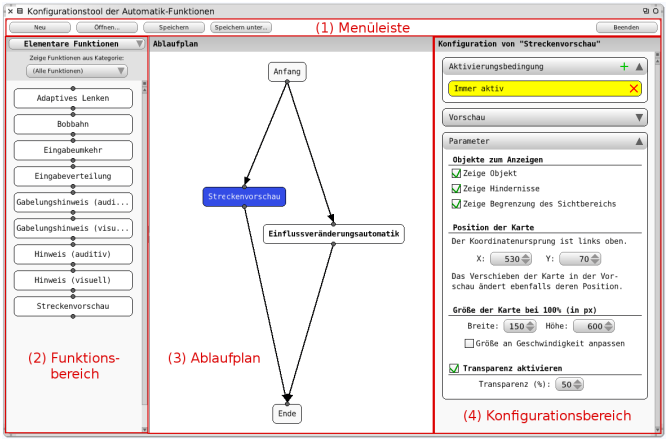


Abbildung C.2: Die Automatiken-GUI in Aktion

1. **Menüleiste mit wichtigen Programmfunktionen (oben)**
   Über die Buttons in der Leiste kann eine neue Automatik erstellt oder geöffnet werden. Eine gerade bearbeitete Automatik kann gespeichert werden. Ein weiterer Button dient zum Beenden des Programms.

2. **Funktionsbereiche für elementare und kombinierte Funktionen (links)**
   Es wird nur ein Funktionsbereich von beiden angezeigt. Über ein Dropdown-Menü kann der gewünschte Bereich ausgewählt werden. Kombinierte Funktionen sind abgespeicherte Automatiken, die aus elementaren oder anderen kombinierten Funktionen bestehen. Kombinierte Funktionen werden in Fettdruck dargestellt.

3. **Ablaufplan (mitte)**
   In diesem Bereich wird eine Automatik strukturell zusammengestellt oder bearbeitet. Die zusammengestellten Funktionen müssen direkt oder indirekt mit dem “Anfang” und dem “Ende” verbunden sein. Zwei Funktionen können entweder parallel oder in Folge zueinander angeordnet werden (Abb. C.2 zeigt eine parallele Anordnung).

4. **Konfigurationsbereich (rechts)**
   Eine angeklickte Funktion im Ablaufplan kann in diesem Bereich konfiguriert werden. Neben der Aktivierungsbedingung (wann soll die Funktion aktiv sein?) können einzelne Parameter der Funktion eingestellt werden.

C.4 AUFBAU EINER AUTOMATIK

C.4.1 *Funktionen hinzufügen*

Funktionen werden zu einer Automatik hinzugefügt, indem sie aus einem der beiden Funktionsbereiche auf der linken Seite in den Ablaufplan im mittleren Bereich der Anwendung gezogen und dort fallen gelassen werden (“Drag and Drop”, siehe Abbildung C.3).

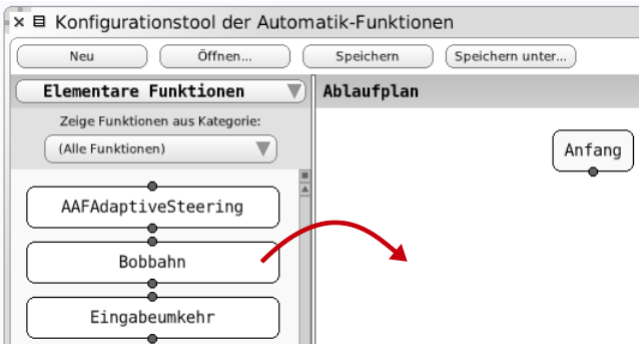


Abbildung C.3: Eine Funktion kann aus dem Funktionsbereich durch anklicken und ziehen dem Ablaufplan hinzugefügt werden.

C.4.2 *Funktionen entfernen*

Soll eine Funktion aus der Automatik entfernt werden, so wird sie mit der rechten Maustaste angeklickt. Im nun erscheinenden Kontextmenü wird der Punkt “Löschen” ausgewählt (s. Abbildung C.4). Ist die Funktion mit anderen Funktionen verbunden, so werden die Verbindungen ebenfalls gelöscht.

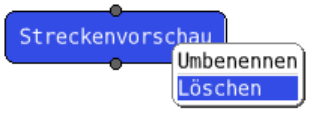


Abbildung C.4: Über das Kontextmenü kann eine Funktion gelöscht werden.

Alternativ lässt sich eine Funktion auch löschen, indem sie “zurück” in den Funktionsbereich gezogen wird. Dabei wird die Anzeige gegebenenfalls zum passenden Funktionsbereich wechseln.

C.4.3 *Verbinden von Funktionen*

Um Funktionen in den Verarbeitungsablauf zu integrieren, müssen diese direkt oder indirekt mit “Anfang” und “Ende” verbun-

den sein. Zum Verbinden von Funktionen wird folgendermaßen vorgegangen:

1. Einen Verbindungspunkt einer Funktionen anklicken. Er ist nun aktiviert und wird orange hervorgehoben (s. Abbildung C.5).

Abbildung C.5: Der erste Verbindungspunkt ist aktiviert.

2. Anschließend den Verbindungspunkt anklicken, mit dem die Verbindung hergestellt werden soll. Die Verbindungslinie wird gezogen (s. Abbildung C.6).

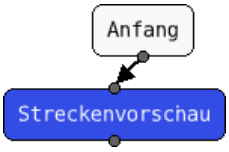


Abbildung C.6: Die Verbindung zwischen zwei Funktionen wurde hergestellt.

Soll ein bereits aktivierter und orange angezeigter Verbindungspunkt doch nicht verbunden werden, so wird er durch erneutes Anklicken deaktiviert.

Beim Ziehen von Verbindungen ist zu beachten, dass nur jeweils ein Eingang mit einem Ausgang verbunden werden kann. Dabei ist es gleichgültig, ob die Verbindung vom Eingang zum Ausgang oder umgekehrt gesetzt wird. Eingang und Ausgang der selben Funktion können nicht verbunden werden.

Jede Funktion mit Ausnahme von Anfang und Ende kann nur eine eingehende und eine ausgehende Verbindung herstellen.

Der Anfang kann beliebig viele ausgehende, das Ende beliebig viele eingehende Verbindungen haben.

C.4.4 *Verbindungen entfernen*

Um eine existierende Verbindung zwischen zwei Funktionen wieder zu löschen, wird die Verbindung mit einem Links-Klick angeklickt und weggezogen (s. Abbildung C.7).

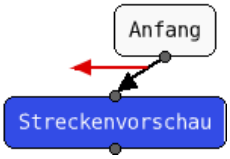


Abbildung C.7: Das Wegziehen von Verbindungen entfernt diese.

C.4.5 *Eine Funktion umbenennen*

Um den Namen einer Funktion zu ändern, wird mit der rechten Maustaste das Kontextmenü aufgerufen und "Umbenennen" ausgewählt. In das erscheinende Texteingabefeld kann nun ein neuer Name eingegeben werden (s. Abbildung C.8).

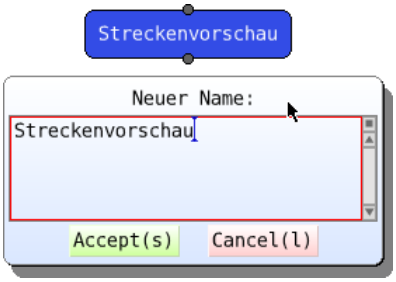


Abbildung C.8: Eine Funktion kann umbenannt werden.

C.4.6 *Bearbeitung der Parameter einer Funktion*

Um die Einstellungen einer Funktion zu bearbeiten, wird die Funktion per Links-Klick im Ablaufplan ausgewählt. Die zugehörigen Einstellungen erscheinen nun im Konfigurationsbereich auf der rechten Seite und können bearbeitet werden (s. Abbildung C.9). Nicht alle Funktionen stellen Parameter zur Verfügung.

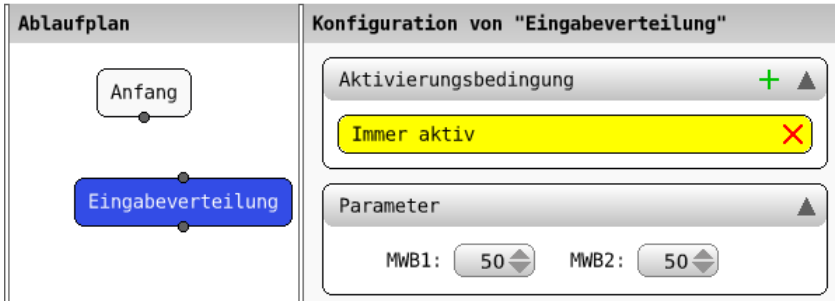


Abbildung C.9: Im Konfigurationsbereich können die Einstellungen einer ausgewählten Funktion bearbeitet werden.

C.4.7 *Festlegen wann eine Funktion aktiv sein soll*

Für jede Funktion kann festgelegt werden, wann diese aktiv sein soll. Dies wird über die "Aktivierung" im Konfigurationsbereich eingestellt (s. Abbildung C.10). Die Aktivierungsbedingung wird durch einen UND/ODER-Ausdruck festgelegt: Die Ereignisse können durch die Operatoren ODER sowie UND verknüpft werden, um die gewünschte Aktivierungsbedingung zu konfigurieren. Die Operanden des Ausdrucks sind dabei Ereignisse wie z. B. 'Kurve voraus', 'Richtungskonflikt festgestellt' oder 'Es wird zu langsam gefahren'.

Abbildung C.10: Die Aktivierungsbedingung wird durch ein UND/ODER-Ausdruck dargestellt, wobei die Operanden konfigurierbare Ereignisse sind.

Das Ereignis 'Immer aktiv' ist die Voreinstellung bei der Aktivierungsbedingung, wenn eine neue Automatik erstellt wird.

Die Ereignisse und Operatoren werden durch gelbe und abgerundete Rechtecke dargestellt. Auf der linken Seite wird der Ereignis- oder Operator-Name dargestellt, auf der rechten Seite stehen Buttons zur Verfügung die in folgenden Unterabschnitten erklärt werden.

C.4.7.1 *Ereignisse und Operatoren hinzufügen*

Durch einen Links-Klick auf das grüne Plus-Symbol in der Kopfzeile des Fensters der Aktivierungsbedingung öffnet sich ein Menü, aus dem die Ereignisse und Operatoren hinzugefügt werden können, die Teil der Aktivierungsbedingung sein sollen (s. Abbildung C.11).

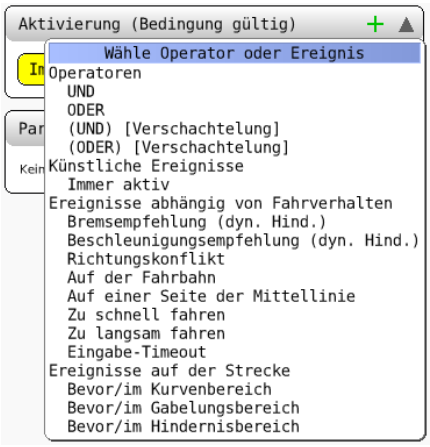


Abbildung C.11: Über das grüne Plus-Symbol kann mit einem Linksklick ein Ereignis oder ein Operator hinzugefügt werden.

Bei den Operatoren kann zwischen den einfachen Operatoren "UND" sowie "ODER" und "(UND) [Verschachtelung]" sowie "(ODER) [Verschachtelung]" ausgewählt werden. Die letzten beiden sind auszuwählen, wenn schon ein einfacher "UND" oder "ODER"- Operator eingefügt wurde. In Abbildung C.10 auf der vorherigen Seite entspricht der obere Operator dem Eintrag "(ODER) [Verschachtelung]" und der untere dem Eintrag "UND". Operatoren mit Klammern sind notwendig, um die Zuordnung der Operanden zu Operatoren eindeutig festzulegen.

C.4.7.2 *Ereignisse und Operatoren entfernen*

Ereignisse oder Operatoren können durch ein Links-Klick auf das rote X aus der Aktivierungsbedingung entfernt werden (s. Abbildung C.10 auf der vorherigen Seite).

C.4.7.3 *Ausdruck bearbeiten durch Verschieben der Ereignisse und Operatoren*

Ein Ereignis oder Operator kann angeklickt und an eine andere Stelle gezogen werden, um eine gültige Aktivierungsbedingung zusammenzustellen. Es empfiehlt sich das gezogene Ereignis direkt über einem anderen Ereignis oder Operator los zu lassen, damit es an dessen Stelle verschoben wird.

C.4.7.4 *Ungültige Ausdrücke erkennen*

Bei einem ungültigen Ausdruck wird "(ungültiger Ausdruck)" in der Kopfzeile des Fensters der Aktivierungsbedingung angezeigt (s. Abbildung C.12).

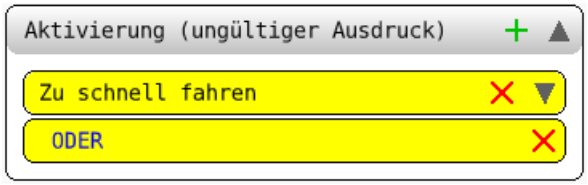


Abbildung C.12: In der Kopfzeile wird bei Erkennung eines ungültigen Ausdrucks gewarnt.

C.4.7.5 *Ereignisse konfigurieren*

Wie die Funktionen können die Ereignisse selbst Parameter zum Konfigurieren bereitstellen. Der Konfigurationsbereich für ein Ereignis kann ausgeklappt werden, in dem mit der linken Maustaste auf das dunkle Dreieck des Ereignisses geklickt wird (s. Abbildung C.13). Durch erneuten Linksklick wird der Konfigurationsbereich wieder eingeklappt.

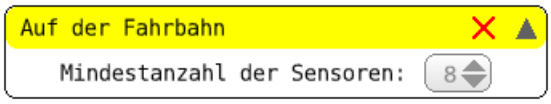


Abbildung C.13: Über das dunkle Dreieck kann der Konfigurationsbereich ein- und ausgeklappt werden.

C.4.8 *Detailgrad erhöhen und verringern*

Kombinierte Funktionen sind an den fett gedruckten Bezeichnern erkennbar (Abbildung C.15 auf der nächsten Seite). Ihr Detailgrad im Ablaufplan kann erhöht werden, um weitere Einstellungen einzusehen und zu verändern.

Ist eine kombinierte Funktion angeklickt, so wird im Konfigurationsbereich ein Button “Detailgrad erhöhen” angezeigt, über den der Ablaufplan der angeklickten, kombinierten Funktion angezeigt werden kann (s. Abbildung C.14). Dieses erscheint dann anstelle des bis dahin sichtbaren Ablaufplans und kann auf die gleiche Weise bearbeitet werden. Alternativ dazu lässt sich der Detailgrad über einen Links-Doppelklick auf die Funktion im Ablaufplan erhöhen. Eine weitere Möglichkeit bietet das Kontextmenü der Funktion.

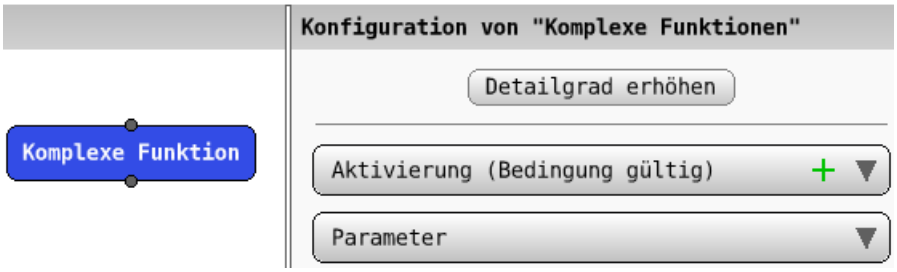


Abbildung C.14: Kombinierte Funktionen haben einen eigenen Ablaufplan. Dieser lässt sich durch den Button "Detailgrad erhöhen" anzeigen und bearbeiten.

Nach der Bearbeitung kann der vorherige Ablaufplan angezeigt werden. Dazu klickt der Benutzer auf den Button "Detailgrad verringern" an der gleichen Stelle im Konfigurationsbereich. Alternativ lässt sich auch hier über das Kontextmenü des Ablaufplans (rechte Maustaste auf freien Bereich innerhalb des Ablaufplans) sowie über einen Doppelklick auf eine freie Fläche des Ablaufplans der Detailgrad verringern.

Wird eine Funktion gespeichert, während ein erhöhter Detailgrad eingestellt ist, so werden alle Funktionen der aktuell bearbeiteten Datei (Siehe Titelleiste des Fensters) gespeichert.

C.5 EINBINDEN BESTEHENDER AUTOMATIKEN

Bereits bestehende Automatiken können als Funktionen neuer Automatiken verwendet werden. Dazu stellt das Automatiken-GUI alle Automatiken, die im Standardverzeichnis für gespeicherte Automatiken liegen und lauffähig sind, im Funktionsbereich für kombinierten Funktionen zur Verfügung (s. Abbildung C.15). Sie können genauso wie elementare Funktionen verwendet werden.

Abbildung C.15: Existierende Automatiken stehen als kombinierte Funktionen in neuen Automatiken zur Verfügung.

C.6 SPEICHERN EINER AUTOMATIK

Um eine Automatik in eine Datei abzuspeichern, wird in der Menüleiste (s. Abbildung C.16) der Button Speichern ausgewählt. Enthält die Automatik Funktionen, die nicht oder nur teilweise mit anderen Funktionen verbunden sind, erscheint eine Warnung die den Benutzer informiert. Fährt der Benutzer mit der Speicherung fort, wird diese Automatik mit dem ungültigen Ablaufplan im Funktionsbereich für kombinierte Funktionen mit einem roten Rahmen hervorgehoben.

Abbildung C.16: Über die Menüleiste sind die Programmfunktionen Neu, Öffnen, Speichern, Speichern unter und Beenden zugänglich.

Über einen Texteingabe-Dialog kann der Benutzer den Namen der Datei angeben. Dabei ist zu beachten, dass der Dateiname auf .aaf enden muss und dass die Speicherung automatisch in das Standardverzeichnis für Automatiken-Dateien [2] erfolgt. Existiert der gewählte Dateiname bereits, so erhält der Benutzer eine entsprechende Warnung und hat die Möglichkeit, einen anderen Namen anzugeben.

Um eine Kopie der aktuellen Datei zu erzeugen und mit dieser weiterzuarbeiten, kann der Button “Speichern unter...” verwendet werden. Im darauf folgenden Texteingabe-Dialog wird nach dem Dateinamen für die Kopie gefragt.

C.7 LADEN EINER AUTOMATIK

Es gibt zwei Möglichkeiten eine früher in eine Datei abgespeicherte Automatik zur erneuten Bearbeitung zu öffnen:

1. Der Button Öffnen in der Menüleiste (Abbildung C.16) wird ausgewählt. Es erscheint ein Auswahldialog, der alle im Standardverzeichnis vorhandenen Automatiken als Dateien zur Auswahl stellt (s. Abbildung C.17).

2 Das Verzeichnis, in dem abgespeicherte Automatiken abgelegt werden, heißt `config.agents`

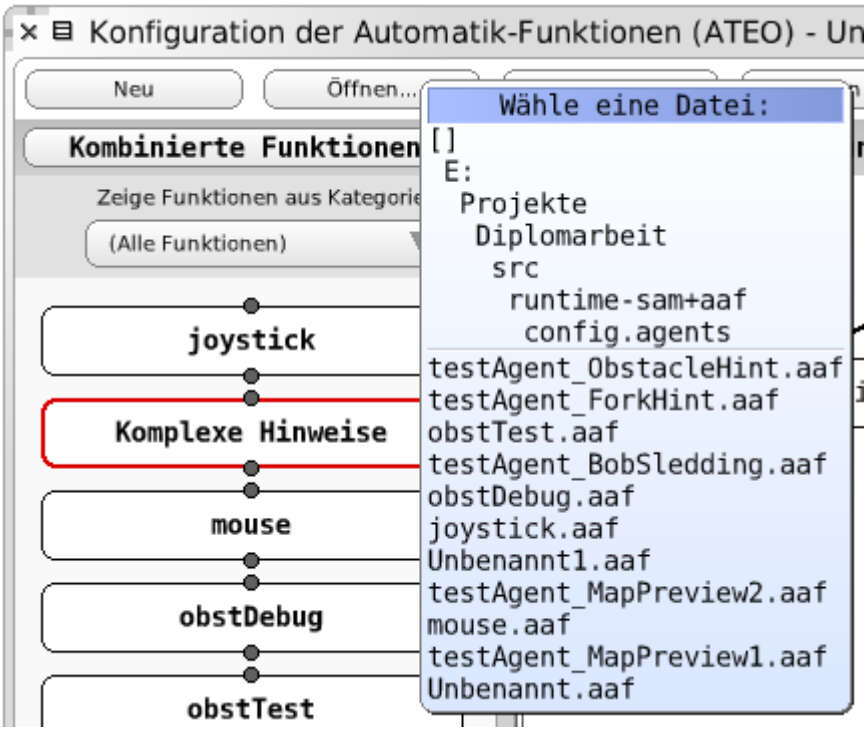


Abbildung C.17: Beim Laden von Automatiken werden die verfügbaren zur Auswahl gestellt.

2. Über das Kontextmenü einer Funktion im Funktionsbereich für kombinierte Funktionen kann Öffnen ausgewählt werden.

C.8 NEUE AUTOMATIK BEGINNEN

Soll eine neue Automatik begonnen werden, so ist der Button Neu in der Menüleiste (Abbildung C.16 auf der vorherigen Seite) zu verwenden. Der Ablaufplan wird geleert und die Bearbeitung kann von Neuem beginnen. Befinden sich in der aktuellen Automatik noch ungespeicherte Änderungen, so erhält der Benutzer beim Klick auf Neu eine entsprechende Warnung sowie die Möglichkeit, diese noch abzuspeichern.

Der Name, die Beschreibung und die Kategorien der neu zu erstellenden Automatik können im Konfigurationsbereich festgelegt werden (s. auch nachfolgender Abschnitt).

C.9 NAME, BESCHREIBUNG UND KATEGORIEN EINER AUTOMATIK FESTLEGEN

Der Name einer Automatik, ihre Beschreibung und die zugeordneten Kategorien können im Konfigurationsbereich bearbeitet werden, wenn im Ablaufplan auf oberster Ebene (geringster Detailgrad) keine Funktion ausgewählt ist (s. Abbildung C.18).

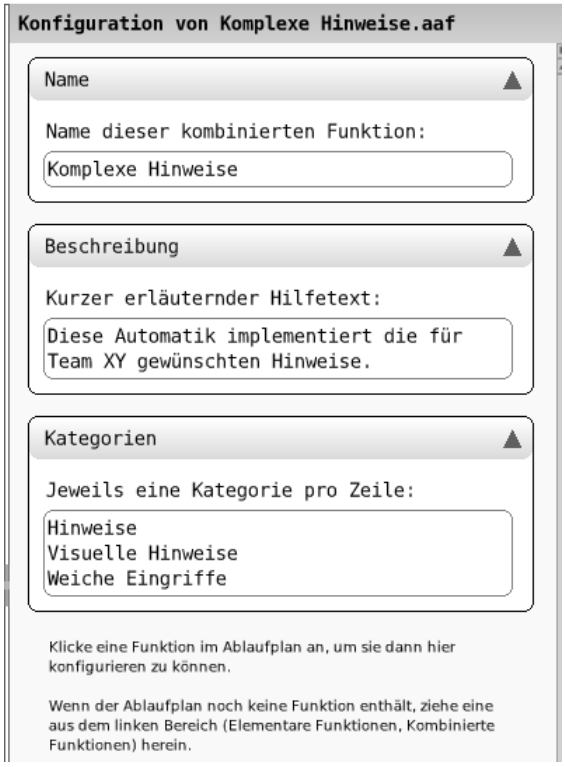


Abbildung C.18: Im Konfigurationsbereich können ein Name, eine Beschreibung sowie verschiedene Kategorien vergeben werden.

Der Name der Automatik bzw. der kombinierten Funktion ist nicht der Dateiname, sondern der Name, wie er im Automatiken-GUI angezeigt wird, wenn abgespeichert wird.

Die Beschreibung soll für den Benutzer einen kleinen Hilfetext zur Verfügung stellen. Sie erscheint als Tooltip, wenn die Maus im Konfigurationsbereich für kombinierte Funktionen über einer Funktion verweilt.

Mit Hilfe von Schlagwörtern können jeder Funktion mehrere Kategorien zugeordnet werden. Die Auswahl vorhandener Automatiken bzw. kombinierter Funktionen kann nach den vergebenen Kategorien gefiltert werden (s. auch nachfolgenden Abschnitt).

C.10 AUSWAHL ELEMENTARER UND KOMBINIERTER FUNKTIONEN

Im linken Bereich des Fensters werden die zur Verfügung stehenden Funktionen angezeigt. Über ein Dropdown-Menü kann zwischen der Anzeige von elementaren und kombinierten Funktionen gewechselt werden. Ein weiteres Dropdown-Menü lässt nur Funktionen einer bestimmten Kategorie anzeigen. Die Kategorien für kombinierte Funktionen können vom Benutzer vergeben werden (vgl. Abbildung C.18).

Im Funktionsbereich für kombinierte Funktionen werden ungültige Funktionen mit einem roten Rahmen hervorgehoben - diese enthalten im Ablaufplan Funktionen, die nicht oder nur teilweise verbunden sind.

C.11 VERLASSEN DES AUTOMATIKEN-GUI

Das Automatiken-GUI kann alternativ über den Beenden-Button in der Menüleiste (Abbildung C.16) oder das Kreuz in der Titelleiste verlassen werden. In beiden Fällen erhält der Benutzer eine Warnung, falls noch nicht gespeicherte Änderungen vorliegen, sowie die Möglichkeit, diese noch abzuspeichern.

C.12 KATALOG DER ELEMENTAREN FUNKTIONEN

In diesem Abschnitt werden die zur Verfügung stehenden elementaren Funktionen und ihre Parameter vorgestellt. Nach dem die gemeinsamen Parameter von auditiven und visuellen Hinweisfunktionen vorgestellt wurden, werden die elementaren Funktionen im Einzelnen dokumentiert.

C.12.1 *Gemeinsame Parameter von auditiven Hinweisfunktionen*

Die Funktionen für auditive Hinweise (betrifft die Funktionen in Abschnitt C.12.3 und C.12.6) verfügen ebenfalls über gleiche Parameter bzw. Widgets. Diese Parameter werden im Folgenden beschrieben, um sie nicht wiederholend in den einzelnen Abschnitten beschreiben zu müssen (siehe Abbildung C.19):

Abbildung C.19: Parameter der Funktion „Hinweis (auditiv)“.

- *Auswählen einer zu abspielenden MP3-Datei*
  Mit einem Links-Klick auf die Grafik kann über ein Menü eine MP3-Datei ausgewählt werden. Per Voreinstellung werden die MP3-Dateien in dem Verzeichnis „resource.sounds“ angezeigt. Über das Menü kann auch ein anderes Quell-

verzeichnis ausgewählt werden, sodass eine beliebige MP3-Datei angegeben werden kann.
Die ausgewählte MP3-Datei wird abgespielt, sobald die Funktion aktiviert wird und die Datei nicht bereits schon abgespielt wird. Es ist also sichergestellt, dass das Abspielen einer Sound-Datei nicht von der gleichen Sound-Datei abgebrochen oder teilweise überlagert wird. Wird die Funktion über mehrere Simulationsschritte aktiviert, kann es passieren, dass die ausgewählte Sound-Datei mehrmals hintereinander abgespielt wird. Falls das nicht erwünscht ist, muss die Aktivierung der Funktion eingeschränkt werden. Für Ereignisse mit Streckenbezug ist das zum Beispiel möglich, in dem der Aktivierungsbereich eingeschränkt wird (siehe Abschnitt C.13.1).

- *Button „Abspielen“*
  Der Button „Abspielen“ dient dem Testen der MP3-Datei. So kann sichergestellt werden, dass die richtige Datei ausgewählt ist (möglicherweise war der Dateiname nicht aussagekräftig genug), diese abspielbar ist und auch die richtige Lautstärke[3] eingestellt ist.

C.12.2 *Gemeinsame Parameter von visuellen Hinweisfunktionen*

Die Funktionen für visuelle Hinweise (betrifft die Funktionen in Abschnitt C.12.4, C.12.5, C.12.7 und C.12.8) verfügen über gleiche Parameter bzw. Widgets. Diese Parameter werden im Folgenden beschrieben, um sie nicht wiederholend in den einzelnen Abschnitten beschreiben zu müssen.

- *Vorschau-Bereich* (siehe Abbildung C.20)
  Dieser Bereich befindet sich immer über dem Parameter-Bereich. Die vorgenommenen Einstellungen im Parameter-Bereich werden in diesem Bereich widergespiegelt. Durch Drag and Drop kann der abgebildete Hinweis wie gewünscht positioniert werden. Die Koordinaten (siehe weiter unten) im Parameter-Bereich werden entsprechend aktualisiert. Je nach Funktion sieht der Vorschau-Bereich jeweils etwas anders aus.

3 Die Lautstärke muss über das Betriebssystem eingestellt werden.

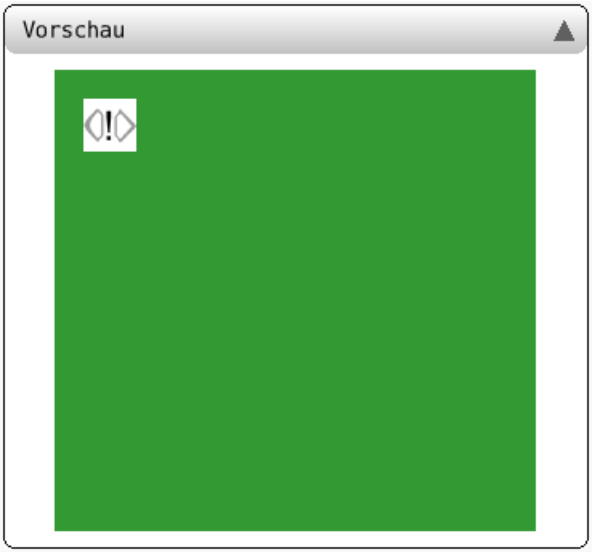


Abbildung C.20: Der Vorschau-Bereich der Funktion „Hinweis (visuell)“.

- *Auswahl eines Bildes* (siehe Abbildung C.21)
  Durch einen Links-Klick auf das aktuell ausgewählte Bild erscheint ein Fenster, in dem ein neues Bild ausgewählt werden kann. Das Quellverzeichnis für die dargestellten Bilder in dem Auswahl-Fenster ist fest mit der Funktion verbunden, sodass kein Bild ausserhalb des voreingestellten Quellverzeichnisses ausgewählt werden kann. Ein neu ausgewähltes Bild wird im Vorschau-Bereich angezeigt.

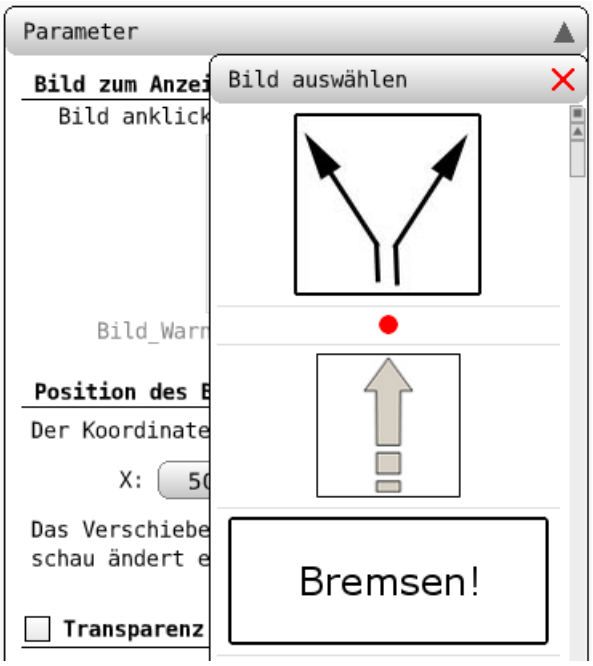


Abbildung C.21: Auswählen eines Bildes bei der Funktion „Hinweis (visuell)“.

- *Angabe von Koordinaten* (siehe Abbildung C.22)
  Ein ausgewähltes Bild kann durch Angabe eines Koordinatenpaares wie gewünscht positioniert werden. Über die Spinboxen kann ein X- und Y-Wert festgelegt werden. Neu eingestellte Werte werden im Vorschau-Bereich widergespiegelt.

Abbildung C.22: Parameter der Funktion „Hinweis (visuell)“.

- *Bestimmung der Transparenz* (siehe Abbildung C.22)
  Durch das Aktivieren der Checkbox „Transparenz aktiv“ wird das ausgewählte Bild im Versuchsschritt (und im Vorschau-Bereich) transparent dargestellt. Die Transparenz kann als Prozent-Angabe festgelegt werden.
- *Bestimmung des Blinkverhaltens* (siehe Abbildung C.22)
  Durch das Aktivieren der Checkbox „Blinken aktiv“ wird das ausgewählte Bild im Versuchsschritt (und im Vorschau-Bereich) blinkend dargestellt. Über die Ein- und Ausblendedauer kann das Blinken angepasst werden.

C.12.3 *Funktion „Hinweis (auditiv)“*

Diese Funktion spielt eine MP3-Datei ab. Diese Funktion kann zum Beispiel genutzt werden, um auditive Warnungen vor bestimmten Streckenelementen zu realisieren.

Abbildung C.23 zeigt die Parameter dieser Funktion. Siehe auch Abschnitt C.12.1 für gemeinsame Parameter von auditiven Hinweisfunktionen.

Abbildung C.23: Parameter der Funktion „Hinweis (auditiv)“.

C.12.4 *Funktion „Hinweis (visuell)“*

Diese Funktion zeigt ein Bild an. Diese Funktion kann zum Beispiel genutzt werden, um visuelle Warnungen vor bestimmten Streckenelementen zu realisieren.

Abbildung C.23 zeigt die Parameter dieser Funktion. Siehe auch Abschnitt C.12.1 für gemeinsame Parameter von auditiven Hinweisfunktionen.

Abbildung C.24: Parameter der Funktion „Hinweis (visuell)“.

Der in Abbildung C.25 gezeigte Vorschau-Bereich spiegelt die Parameter der Funktion wider.

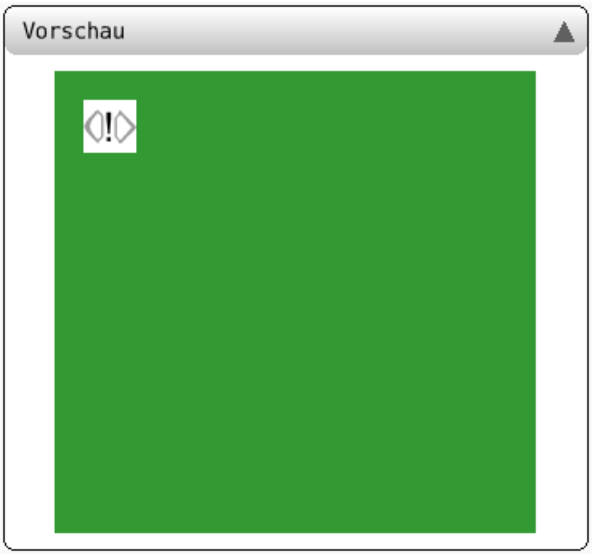


Abbildung C.25: Der Vorschau-Bereich der Funktion „Hinweis (visuell)“.

C.12.5 *Funktion „Hindernisslalom (visuell)“*

Die Funktion zeigt ein Bild im Bereich eines Hindernis-Slaloms an. Per Voreinstellung bringt diese Funktion ein dafür günstig konfiguriertes Ereignis „Bevor/im Hindernisbereich“ mit. Die Aktivierung dieser Funktion muss daher nicht angepasst werden. Diese Funktion kann z. B. verwendet werden, um ein Pfad zum Umfahren des Hindernis-Slaloms anzuzeigen.

Abbildung C.26 zeigt die Parameter dieser Funktion. Über das Drop-Down-Menü kann zwischen einem Hindernis-Slalom mit „25%-Fahrbahnüberdeckung“ und „50%-Fahrbahnüberdeckung“ ausgewählt werden. Bei Auswahl wird der Vorschau-Bereich entsprechend aktualisiert.

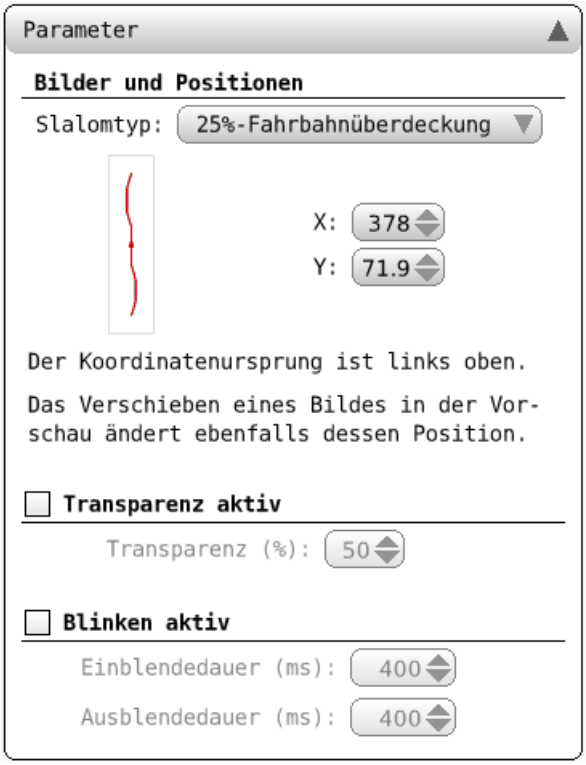


Abbildung C.26: Parameter der Funktion „Hindernisslalom (visuell)“.

Der in Abbildung C.27 gezeigte Vorschau-Bereich spiegelt die Parameter der Funktion wider.

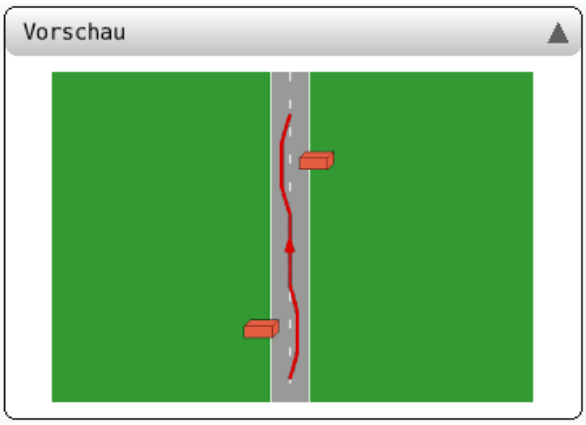


Abbildung C.27: Der Vorschau-Bereich der Funktion „Hindernisslalom (visuell)“.

C.12.6 *Funktion „Gabelungshinweis (auditiv)“*

Mit dieser Funktion kann den Mikroweltbewohnern eine auditive Zweigempfehlung ausgesprochen werden. Per Voreinstellung bringt diese Funktion ein dafür günstig konfiguriertes Ereignis „Bevor/im Gabelungsbereich“ mit. Die Aktivierung dieser Funktion muss daher nicht angepasst werden.

Über das Drop-Down-Menü in Abbildung C.28 kann eingestellt werden, welcher Zweig empfohlen werden soll. Es stehen folgende Vorschlagstypen zur Auswahl: „Besserer/Optimaler Zweig“, „Dünner Zweig“, „Breiter Zweig“, „Kurzer Zweig“, „Langer Zweig“, „Schneller Zweig“, „Rechter Zweig“, „Einfacher Zweig“, „Schwieriger Zweig“ und „Zufälliger Zweig“. Der Vorschlagstyp „Besserer/Optimaler Zweig“ entspricht zur Zeit dem Vorschlagstyp „Schneller Zweig“.

Für die Zweigempfehlung kann gesondert für den linken und rechten Zweig eine MP3-Datei zum Abspielen ausgewählt werden. Wenn als Vorschlagstyp „Rechter Zweig“ ausgewählt wird, dann ist es unnötig eine MP3-Datei für den linken Zweig festzulegen, da diese im Versuchsschritt nicht abgespielt wird.

Siehe auch Abschnitt C.12.1 für gemeinsame Parameter von auditiven Hinweisfunktionen.

Abbildung C.28: Parameter der Funktion „Gabelungshinweis (auditiv)“.

C.12.7 *Funktion „Gabelungshinweis (visuell)“*

Mit dieser Funktion kann den Mikroweltbewohnern eine visuelle Zweigempfehlung ausgesprochen werden. Per Voreinstellung

bringt diese Funktion ein dafür günstig konfiguriertes Ereignis „Bevor/im Gabelungsbereich“ mit. Die Aktivierung dieser Funktion muss daher nicht angepasst werden.

Das Drop-Down-Menü in Abbildung C.29 ist in Abschnitt C.12.6 beschrieben.

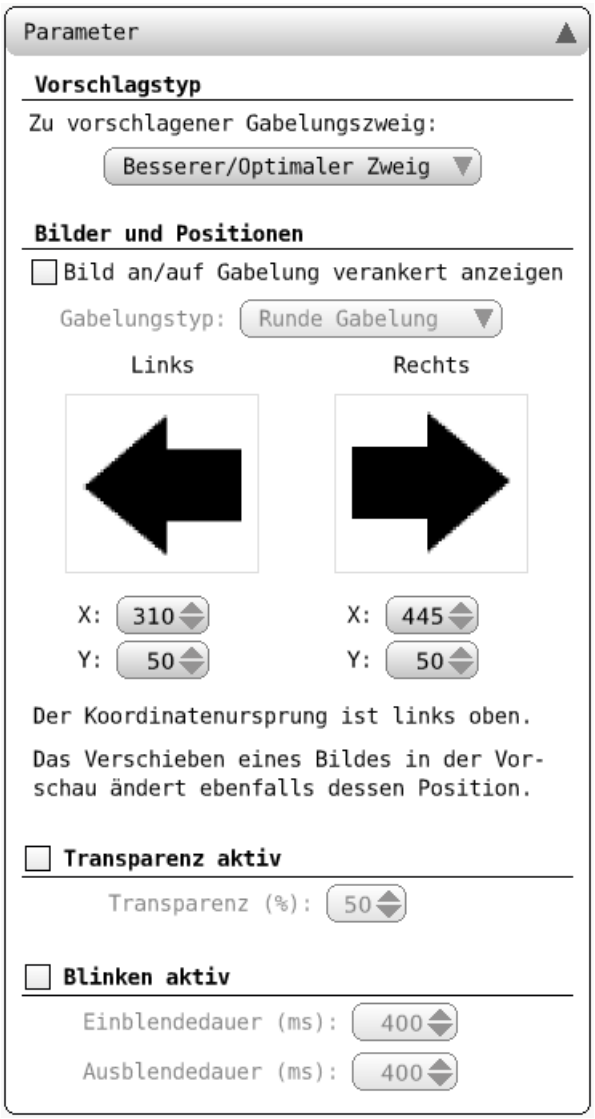


Abbildung C.29: Parameter der Funktion „Gabelungshinweis (visuell)“.

Für die Zweigempfehlung kann gesondert für den linken und rechten Zweig ein Bild ausgewählt werden. Wenn als Vorschlagstyp „Rechter Zweig“ ausgewählt wird, dann ist es unnötig ein Bild für den linken Zweig festzulegen, da dieses im Versuchsschritt nicht angezeigt werden wird.

Diese Funktion unterstützt zwei Möglichkeiten die Bilder einzublenden.

- *„Bild an/auf Gabelung verankert anzeigen“ ist deaktiviert*
  Die Bilder werden an festen Positionen im Sichtbereich an-

gezeigt. Dabei läuft die Strecke „unter“ dem angezeigten Bild mit fortlaufenden Simulationsschritten weiter.
Der Vorschau-Bereich sieht in diesem Fall wie in Abbildung C.30 gezeigt aus.

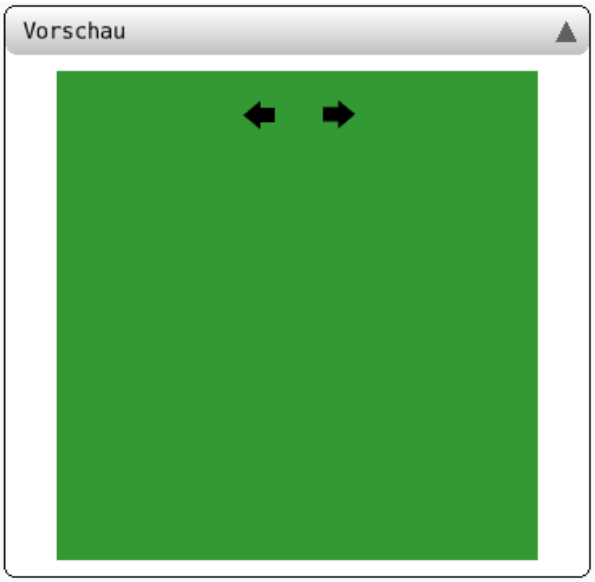


Abbildung C.30: Der Vorschau-Bereich der Funktion „Gabelungshinweis (visuell)“.

- *„Bild an/auf Gabelung verankert anzeigen“ ist aktiviert*
  Die Bilder werden auf der Strecken (an den Gabelungen) „verankert“ und laufen zusammen mit der Strecke mit. Der Vorschau-Bereich sieht in diesem Fall z. B. wie in Abbildung C.30 gezeigt aus. Das Drop-Down-Menü bei „Gabelungstyp:“ ist dann aktiviert und es kann entweder die runde oder eckige Gabelung ausgewählt werden, sodass die Bilder und Positionen gesondert für diese Gabelungstypen eingestellt werden können. Der ausgewählte Gabelungstyp wird im Vorschau-Bereich angezeigt.

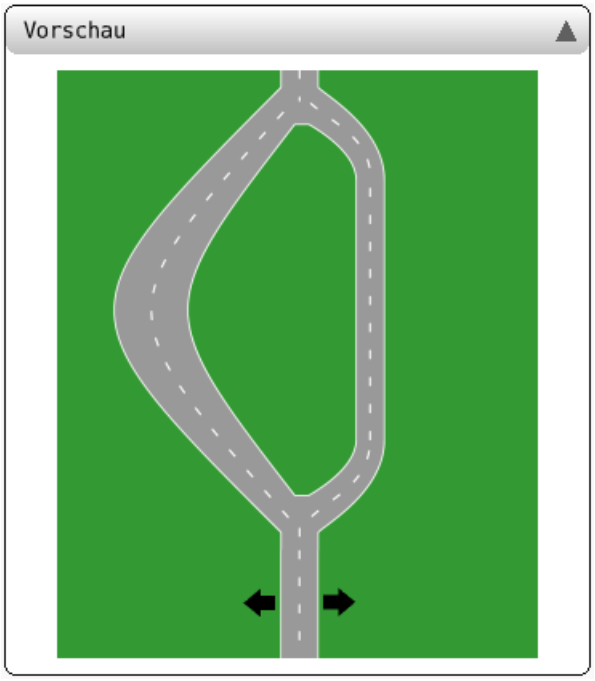


Abbildung C.31: Der Vorschau-Bereich der Funktion „Gabelungshinweis (visuell)“.

Im Vorschau-Bereich werden stets die beiden ausgewählten Bilder angezeigt. Diese können auch getrennt von einander via Drag and Drop positioniert werden. Während des Versuchsschritts wird jedoch nur ein Bild angezeigt, da nur ein Zweig empfohlen wird.

Siehe auch Abschnitt C.12.2 für gemeinsame Parameter von visuellen Hinweisfunktionen.

C.12.8 *Funktion „Streckenvorschau (visuell)“*

Diese Funktion blendet den aktuell sichtbaren Teil der Strecke sowie Teile der zukünftigen Strecke verkleinert ein.

Die Streckenvorschau kann mit den Parameter in Abbildung C.32 eingestellt werden. Teil der Streckenvorschau sind das Objekt, die Hindernisse sowie der horizontale Strich (in Abbildung C.33 auf Seite 168 direkt unter dem Hindernis zu sehen), der die Begrenzung des aktuellen Sichtbereichs markiert.

Weitere Parameter sind die Position auf der Streckenvorschau im Sichtbereich, die Größe der Streckenvorschau sowie die Transparenz. Mit dem Parameter „Größe an Geschwindigkeit anpassen“, wird die Größe der Streckenvorschau ab 75% der maximalen Geschwindigkeit gleitend auf bis zu 70% ihrer ursprünglichen Größe reduziert. Damit wird bei hohen Geschwindigkeiten den Mikroweltbewohnern eine hinreichend weite Vorschau ermöglicht.

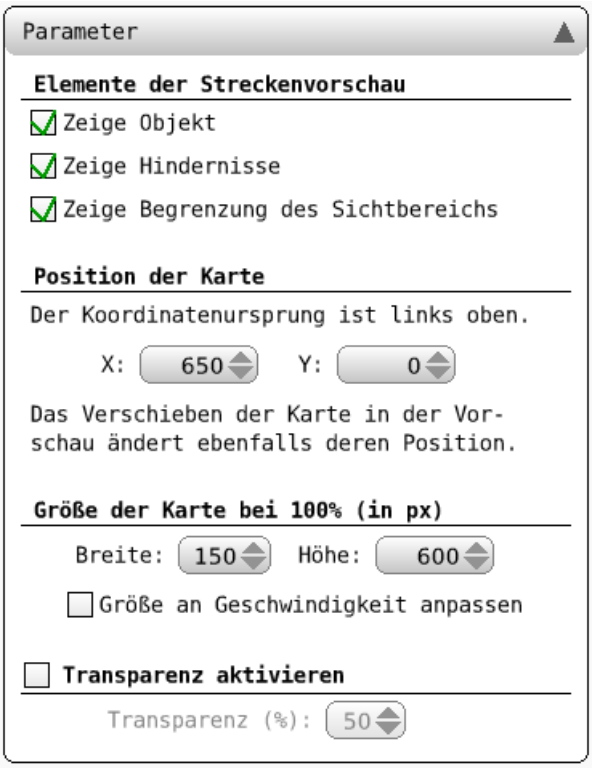


Abbildung C.32: Parameter der Funktion „Streckenvorschau (visuell)“.

Der in Abbildung C.27 gezeigte Vorschau-Bereich spiegelt die Parameter der Funktion wider. Ist „Größe an Geschwindigkeit anpassen“ aktiviert, so kann die Größenanpassung der Streckenvorschau auch im Vorschau-Bereich nachvollzogen werden. Dazu kann mit dem Spinbutton die Geschwindigkeit eingestellt werden. Ab 76% der maximalen Geschwindigkeit wird die Größenskalierung aktiv.

Abbildung C.33: Der Vorschau-Bereich der Funktion „Streckenvorschau (visuell)“

Siehe auch Abschnitt C.12.2 für gemeinsame Parameter von visuellen Hinweisfunktionen.

C.13 KATALOG DER EREIGNISSE

In diesem Abschnitt werden die zur Verfügung stehenden Ereignisse und ihre Parameter vorgestellt. Wie in Abschnitt C.4.7.1 vorgestellt, können die Ereignisse über das grüne Plus-Symbol im Fenster „Aktivierung“ hinzugefügt werden. Befindet sich auf der rechten Seite des hinzugefügten Ereignisses ein Dreieck, können durch einen Links-Klick auf dieses die zugehörigen Parameter aufgeklappt werden. Durch erneuten Links-Klick auf das Dreieck werden die Parameter wieder zugeklappt.

C.13.1 *Gemeinsame Parameter von Ereignissen mit Streckenbezug*

Die Ereignisse mit Streckenbezug (betrifft die Ereignisse in Abschnitt C.13.2, C.13.3 und C.13.4) verfügen teilweise über die gleichen Parameter. Diese Parameter werden im Folgenden beschrieben, um sie nicht wiederholend in den einzelnen Abschnitten beschreiben zu müssen.

Für ein Ereignis mit Streckenbezug kann festgelegt werden, in welchem Bereich des Streckenelementes dieses aktiv sein soll. Das sei an folgendem Beispiel erklärt. Im unteren Bereich der Abbildung C.34 sind die relevanten Parameter zu sehen. Die

Semantik ist: Wenn sich das Objekt 300 Pixel vor[4] dem Anfang[5] der Gabelung (obere Spinbox in der Abbildung) und 300 Pixel nach dem Anfang der Gabelung befindet (untere Spinbox in der Abbildung), dann ist das Ereignis aktiv. Anderenfalls ist das Ereignis nicht aktiv.

Es sei angemerkt, dass der eingestellte Abstand nach einem Streckenelement-Anfang (untere Spinbox) größer sein kann, als das Streckenelement hoch ist. Das Ereignis kann aber maximal nur so lange eintreten, wie das Objekt noch vor dem Ende[6] des Streckenelements ist. Beispiel: Die Höhe einer Gabelung beträgt 800 Pixel. Wird in der unteren Spinbox ein Wert von 1200 eingestellt, so verhält sich das Ereignis nicht anders, als wenn 800 eingestellt wäre, da die Höhe des Streckenelements den tatsächlichen Wert stets limitiert.

C.13.2 *Ereignis „Bevor/im Gabelungsbereich“*

Dieses Ereignis tritt bei Gabelungen ein und verfügt über die in Abbildung C.34 dargestellten Parameter.

Über die Gabelungstypen kann unterschieden werden, ob das Ereignis nur bei runden Gabelungen, nur bei eckigen Gabelungen oder bei beiden Gabelungstypen aktiv sein soll. Wenn keiner der beiden Gabelungstypen selektiert wird, wird das Ereignis nie eintreten.

Die Aktivierung kann weiter auf einen bestimmten Bereich eingeschränkt werden (siehe Abschnitt C.13.1).

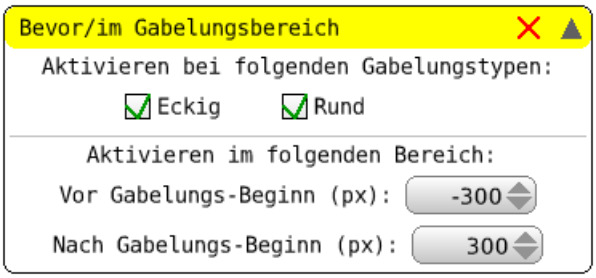


Abbildung C.34: Parameter des Ereignisses „Bevor/im Gabelungsbereich“.

C.13.3 *Ereignis „Bevor/im Hindernisbereich“*

Dieses Ereignis tritt bei Hindernissen ein und verfügt über die in Abbildung C.34 und C.36 dargestellten Parameter.

4 Der Abstand vor dem Streckenelement wird durch eine negative Zahl angegeben.
5 Mit dem Anfang ist die erste Pixelzeile des entsprechenden Bildes gemeint.
6 Mit dem Ende ist die letzte Pixelzeile des entsprechenden Bildes gemeint.

Über das Drop-Down-Menü oben kann eingestellt werden, ob das Ereignis nur für bestimmte, einzelne Hindernisse oder bei Hindernis-Slaloms eintreten soll:

- „Einzelnes Hindernis“ (siehe Abbildung C.13.3)
  Die Auswahl des „Einzelnes Hindernis“ bietet sich an, wenn nur ein bestimmtes Hindernis die Aktivierung auslösen soll. Unter dem Drop-Down-Menü kann spezifiziert werden, welche Hindernisse von Interesse sind. Das Ereignis tritt zu allen aktivierten Hindernistypen ein. Wenn kein Hindernistyp ausgewählt wird, wird das Ereignis nie eintreten.

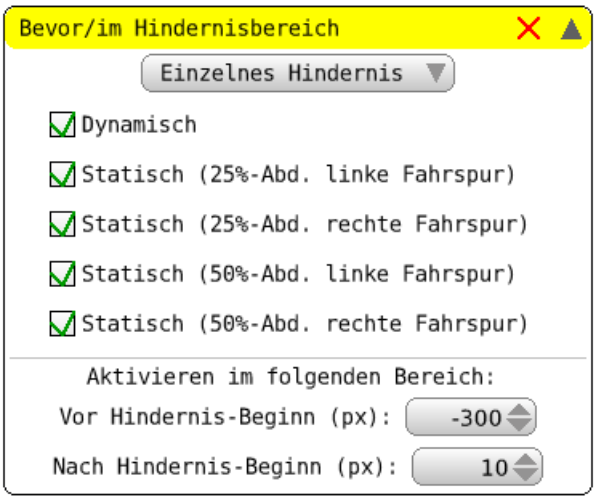


Abbildung C.35: Parameter des Ereignisses „Bevor/im Hindernisbereich“ (Auswahl „Einzelnes Hindernis“).

- „Hindernis-Slalom“ (siehe Abbildung C.36).
  Die Auswahl eines Hindernis-Slaloms bietet sich an, wenn das Ereignis zwischen dem ersten und zweiten Hindernisses des Slaloms stets aktiv sein soll.

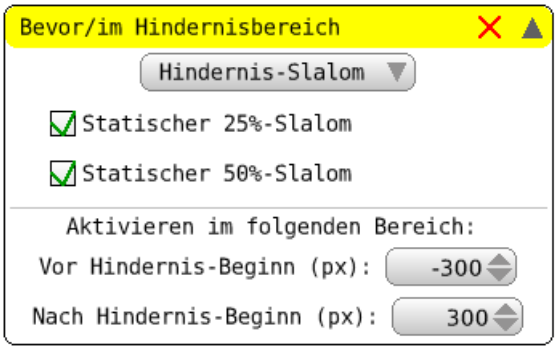


Abbildung C.36: Parameter des Ereignisses „Bevor/im Hindernisbereich“ (Auswahl „Hindernis Slalom“).

Unabhängig von der Drop-Down-Menü-Auswahl, kann die Aktivierung auf einen bestimmten Bereich eingeschränkt werden (siehe Abschnitt C.13.1).

C.13.4 *Ereignis „Bevor/im Kurvenbereich“*

Dieses Ereignis tritt bei Kurven ein und verfügt über die in Abbildung C.34 dargestellten Parameter.

Die gewünschten Kurven, bei denen das Ereignis aktiv sein soll, können durch die Checkboxen bestimmt werden. Das Ereignis tritt für alle ausgewählten Kurventypen ein. Wenn keine Kurventypen ausgewählt werden, wird das Ereignis nie eintreten.

„Kurz“, „Mittel“, „Lang“ und „Sehr lang“ bezieht sich auf die Kurvenbreite. Bei einer kurzen Kurve muss nur für eine kurze Zeit in die entsprechende Richtung gelenkt werden, um weiterhin auf der Fahrbahn zu bleiben. Bei sehr langen Kurven muss entsprechend länger in eine Richtung gelenkt werden.

Es sei daraufhin gewiesen, dass Links- und Rechtkurven aus Sicht der Mikroweltbewohner zu sehen sind und nicht aus einem fiktiven Fahrer aus „innerhalb“ des Objekts[7]. Wenn die Mikroweltbewohner nach links lenken müssen, um den Verlauf der Fahrbahn zu folgen, dann handelt es sich um eine Linkskurve. Analog für Rechtskurven.

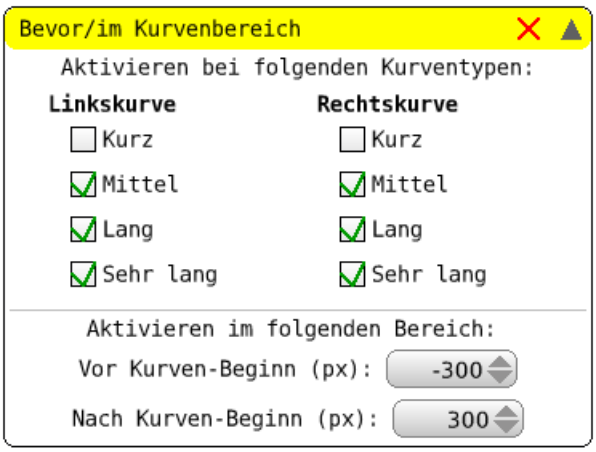


Abbildung C.37: Parameter des Ereignisses „Bevor/im Kurvenbereich“
.

Die Aktivierung kann weiter auf einen bestimmten Bereich eingeschränkt werden (siehe Abschnitt C.13.1).

7 Der Blickwinkel bestimmt die Interpretation einer Links- und Rechtskurve.

C.13.5 *Ereignis „Bremsempfehlung (dyn. Hindernis)“*

Dieses Ereignis tritt ein, wenn sich das Objekt vor dynamischen Hindernissen auf Kollisionskurs befindet und durch Abbremsen des Objekts die Kollision verhindert werden kann. Es kann z. B. sinnvoll mit der Funktion „Hinweis (visuell)“ (Abschnitt C.12.4) genutzt werden.

C.13.6 *Ereignis „Beschleunigungsempfehlung (dyn. Hindernis)“*

Dieses Ereignis tritt ein, wenn sich das Objekt vor dynamischen Hindernissen auf Kollisionskurs befindet und durch Beschleunigung des Objekts die Kollision verhindert werden kann. Es kann z. B. sinnvoll mit der Funktion „Hinweis (visuell)“ (Abschnitt C.12.4) genutzt werden.

C.13.7 *Ereignis „Zu schnell fahren“*

Für jede beliebige Position des Objekts auf der Mittellinie der Fahrbahn ist die ideale Geschwindigkeit bekannt. Dieses Ereignis tritt ein, wenn das Objekt mit höherer Geschwindigkeit als der idealen Geschwindigkeit auf der Mittellinie bewegt wird.

Der in Abbildung C.38 gezeigte Parameter erlaubt es, eine Toleranz zu definieren. Wird der Wert null eingestellt, tritt das Ereignis immer ein, sobald die Mikroweltbewohner das Objekt schneller als mit idealer Geschwindigkeit fahren. Wird der Wert auf zwei eingestellt, wie in der Abbildung, dann tritt das Ereignis erst ein, wenn die Mikroweltbewohner das Objekt schneller als ideale Geschwindigkeit + 2 bewegen. Es wird empfohlen eine Toleranz größer als zwei zu setzen, wenn die Toleranz von null für ein zu häufiges Eintreten des Ereignisses sorgt.

Da bei diesem Ereignis davon ausgegangen wird, dass das Objekt auf der Mittellinie bewegt wird, ist es nur sinnvoll dieses Ereignis zu benutzen, wenn sichergestellt werden kann, dass das Objekt tatsächlich auf der Mittellinie bzw. nahe dieser bewegt wird. Es wird daher empfohlen dieses Ereignis stets mit dem Ereignis „Auf der Fahrbahn“ (siehe Abschnitt C.13.9) zu verwenden (UND-verknüpft).

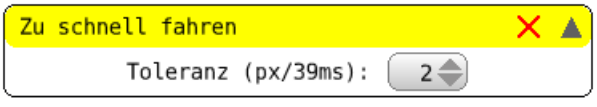


Abbildung C.38: Parameter des Ereignisses „Zu schnell fahren“.

C.13.8 *Ereignis „Zu langsam fahren“*

Analog zu dem Ereignis „Zu schnell fahren“ (siehe Abschnitt 6.2.4). Wird der Toleranzwert auf 2 eingestellt, dann tritt das Ereignis erst ein, wenn die Mikroweltbewohner das Objekt langsamer als ideale Geschwindigkeit - 2 bewegen.

C.13.9 *Ereignis “Auf der Fahrbahn”*

Um das Objekt herum sind acht Sensoren angeordnet die bestimmen können, ob das Objekt auf der Fahrbahn bewegt wird oder nicht. Wird das Objekt auf der Fahrbahn bewegt, dann tritt dieses Ereignis ein.

Mit dem in Abbildung C.39 gezeigten Parameter kann bestimmt werden, wie viele Sensoren melden müssen, dass das Objekt auf der Fahrbahn ist, bevor das Ereignis eintritt. Wird ein Wert von acht eingestellt, wie auf der Abbildung zu sehen, dann tritt das Ereignis immer nur dann ein, wenn das Objekt vollständig auf der Fahrbahn ist (d. h. stets zwischen den Fahrbahnrändern). Der Fahrbahnrand wird als der Fahrbahn zugehörig angesehen.

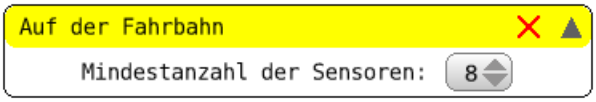


Abbildung C.39: Parameter des Ereignisses „Auf der Fahrbahn“.

C.13.10 *Ereignis „Eingabe-Timeout“*

Dieses Ereignis tritt ein, wenn von einem oder beiden Joysticks für längere Zeit nur Daten kommen, die die Ruhestellung der Joysticks signalisieren. Auf diese Weise kann ein „Personenausfall“ entdeckt werden.

Über das Drop-Down-Menü in Abbildung C.40 kann eingestellt werden, ob die Joysticks von beiden Mikroweltbewohnern oder nur von einem der beiden überwacht werden. Der Timeout legt fest, wie lange Joystick-Daten mit Ruhestellung akzeptiert werden, bevor das Ereignis eintritt.

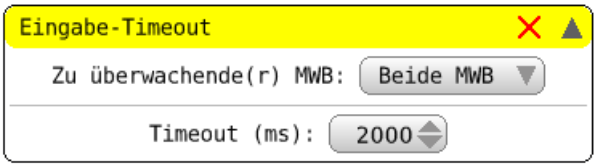


Abbildung C.40: Parameter des Ereignisses „Eingabe-Timeout“

C.13.11 *Ereignis „Auf einer Seite der Mittellinie“*

Dieses Ereignis tritt ein, wenn sich das Objekt auf einer gewünschten Seite der Mittellinie befindet.

Über den in Abbildung C.41 gezeigten Parameter kann die linke oder rechte Seite ausgewählt werden. Für die Bestimmung auf welcher Seite sich das Objekt befindet, wird der Mittelpunkt des Objekts benutzt. Die Mittellinie gehört zu der spezifizierten Seite.

Abbildung C.41: Parameter des Ereignisses „Auf einer Seite der Mittellinie“

D

INHALT DER DVD

Diese Arbeit wird auch in elektronischer Form als DVD eingereicht. Auf der DVD befinden sich:

1. Datei „Diplomarbeit_NikolaiKosjar.pdf“
   Elektronische Fassung dieser Arbeit

2. Verzeichnis „Squeak“
   Quelltexte und ein lauffähiges Squeak-Image

3. Verzeichnis „Usability Untersuchung“ (siehe Abschnitt 7.4)
   Videoaufzeichnungen, Morae-Projektdateien sowie die benutzte Version der Bedienungsanleitung und des Squeak-Images

4. Verzeichnis „Transkriptionen“ (siehe Kapitel 4)
   Transkriptionen der Teams 22, 30, 34 und 38

SELBSTSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 21. März 2012

Nikolai Kosjar