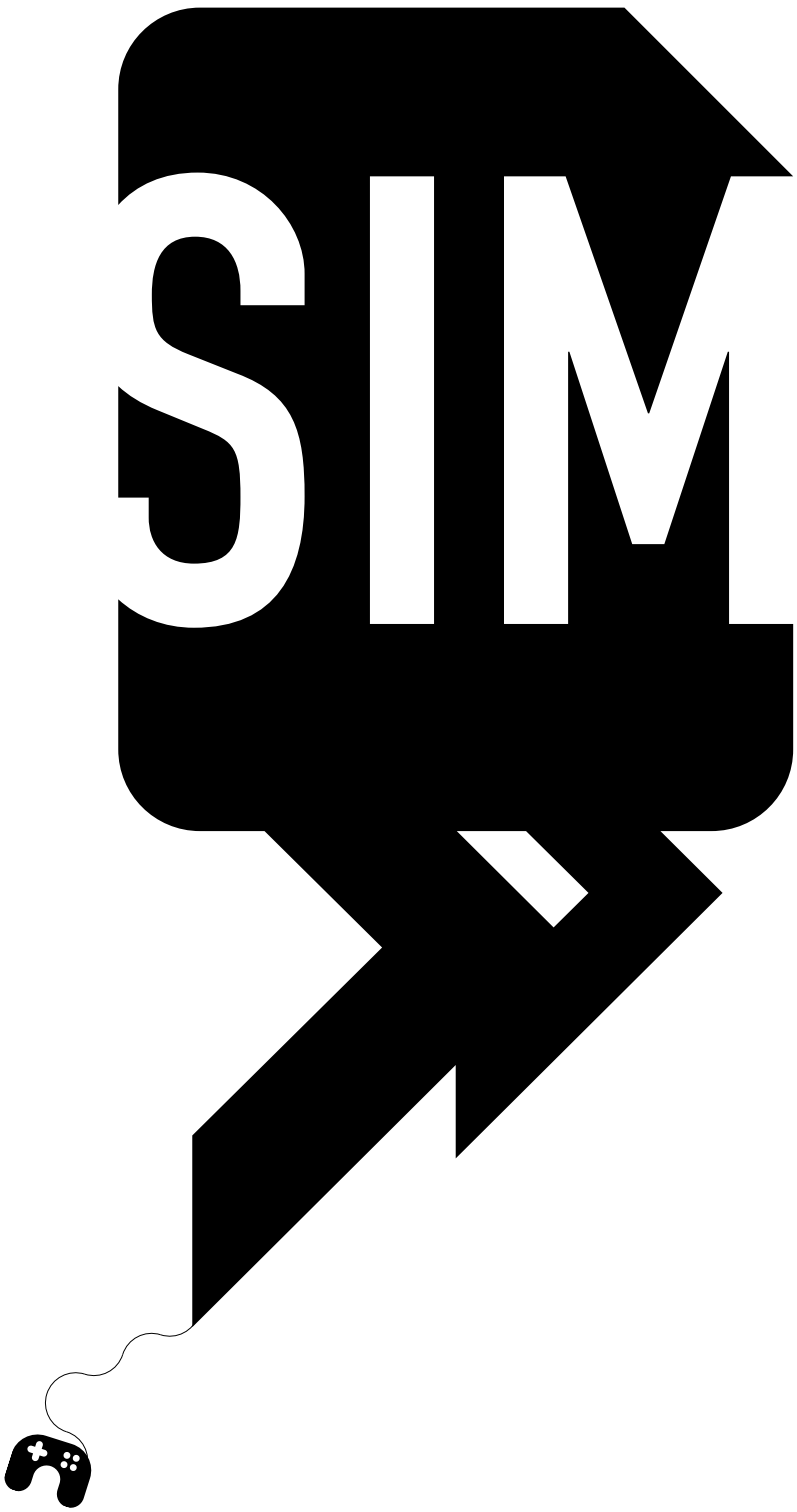

SIM

Untersuchung zur Darstellung von Informationen Durch Verhalten
Anhand von Mobiltelefonbüchern

//////////////////

Dokumentation der Diplomarbeit
von Dennis Paul

Betreut durch Professor Joachim Sauter,
Fachklasse für Gestalten mit Digitalen Medien
Universität der Künste Berlin

Berlin im April 2003

STRUKTUR

00
EINLEITUNG

INFORMATIONSDESIGN, COMPUTATIONALDESIGN, COMPUTERSPIELE,
CORPORATEDESIGN, NATUR, TECHNIK, ALLTAGSDATEN & ICH

ICH INFORMATIONSDESIGN, COMPUTATIONALDESIGN, COMPUTERSPIELE, CORPORATEDESIGN, NATUR, TECHNIK, ALLTAGSDATEN & ICH

Mein Ziel mit dieser Arbeit ist nicht nur, das später beschriebene Thema zu bearbeiten, sondern auch meine Interessen als Gestalter, in dem Spannungsfeld aus den in der Überschrift genannten Polen, einzubeziehen. Deshalb ist die Grundlage meiner Arbeit nicht nur die Visualisierung von sozialen Netzen durch Verhalten, sondern auch das Beziehen einer Position irgendwo in dem beschriebene Spannungsfeld.

AUFSTELLEN VON REGELN UND DIESE DANN BEFOLGEN; DAS IST GESTALTUNG.

Auf der Suche nach einem ›Roten Faden‹ in meiner Haltung zur Gestaltung, ist dieser Satz eine Zustandsbeschreibung. Ich hab mich immer wieder gefragt, warum ich nach einer typographischen Grundausbildung, vom Papier zum Digitalen gewechselt bin. Eine mögliche Antwort ist, dass ich Gestaltung sowieso schon immer als ein weitestgehend programmierbares System verstanden habe (›Max und Milli‹ Plakatreihe /// ›IMS‹ Erscheinungsbild /// ›A Grand Day Out‹) und der Wechsel zu programmierbaren Medien nur eine nächster Schritt in die gleiche Richtung war.

SYSTEM / MODUL / ELEMENT / GENSTRUKTUR / REGELN

Diese Diplomarbeit steht sowohl visuell als auch inhaltlich klar in der Tradition dieses Gestaltungsansatzes.

Enjoy!

ANLIEGEN MEINE ARBEIT SOLL SICH MIT DEM THEMA ›UNTERSUCHUNG ZUR DARSTELLUNG VON INFORMATIONEN DURCH VERHALTEN‹ BEFASSEN.

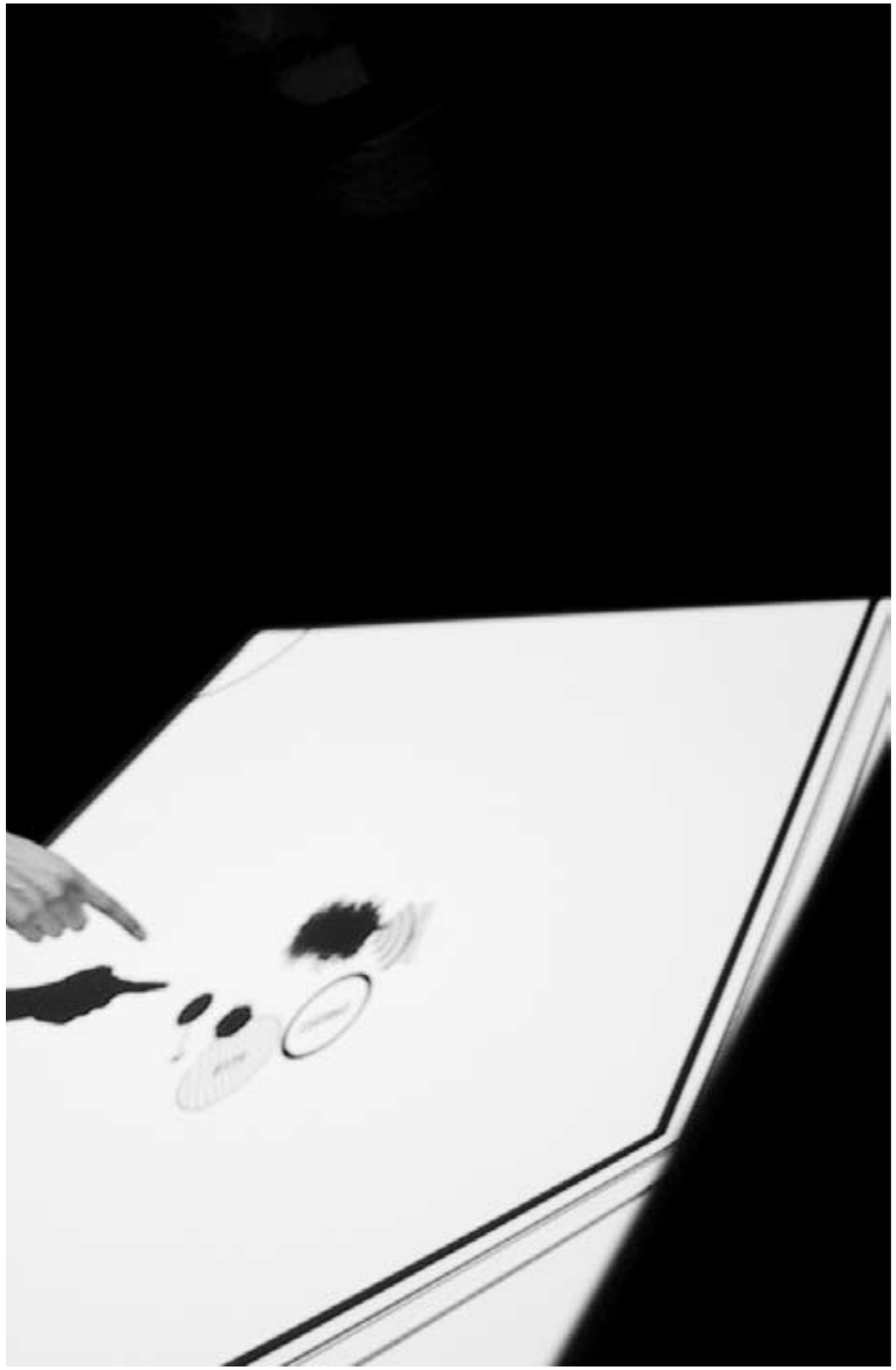
Ich untersuche die Möglichkeiten einer solchen Darstellung am Beispiel von Mobiltelefonbüchern & den sozialen Verknüpfungen, die aus dem Vergleich dieser Telefonbücher ersichtlich werden.

Ich sehe es als einen wesentlichen Bestandteil meiner Arbeit als Gestalter an, mich mit der Erforschung neuer Darstellungsformen zu beschäftigen. Deshalb möchte ich in meiner Arbeit an der Grenze des legitimen Informationsdesigns ein wenig zerren und ziehen. An dieser Stelle muss ich die Arbeit von ›asymtode‹ erwähnen, die, obwohl ich sie nicht unter ›Einflussreiche Arbeiten‹ erwähnt habe, doch in sofern ein Vorbild ist, als dass ihr Umgang mit der Informationsgrafik sehr heftig diskutiert ist, da sie, so die Kritiker, die Lesbarkeit von Informationen zugunsten einer Ästhetisierung der Darstellung aufgeben. Die Ablesbarkeit von Informationen soll in meiner Arbeit nicht mutwillig herabgesetzt werden, ich möchte sie nur verschieben. Ein grosse Stärke der computerbasierten Informationsgestaltung ist die Darstellung von sehr grossen Datenmengen, meistens durch ein unschärfe in der Darstellung, die es ermöglicht ein Gesamtbild zu bekommen oder globale Zusammenhänge von oder in Datenmengen zu sehen. Ich untersuche nun die Möglichkeit dieses Prinzip des unschärfens durch Verhalten zu erzeugen, welches in Form von Bewegung kombiniert mit einem Kontext, im Betrachter bestimmte Assoziationen oder Schlüsse proviziert, die er auf Grund von gelernten oder angeboren Mustern im Kopf hat. Ein Beispiel ist das Kindchenschema, das zB durch einen, im Vergleich zum übrigen Körper, grossen Kopf, in vielen höheren Säugetieren, also auch Menschen, erzeugt wird. Jenachdem in welchem Kontext dieses Kindchenschema benutzt wird, kann es bei dem Betrachter verschiedene Assoziationen hervorrufen; beschützen wollen, etwas ist noch sehr jung, hier hat Fortpflanzung stattgefunden. Ein anderes Beispiel ist schnelle Bewegung von Objekten, die je nach Kontext, zB als freudige Erregung, also bekannte Daten, interpretiert werden kann oder als Erschrecken, was bedeuten kann, dass sich ungleiche Daten in der Umgebung befinden.

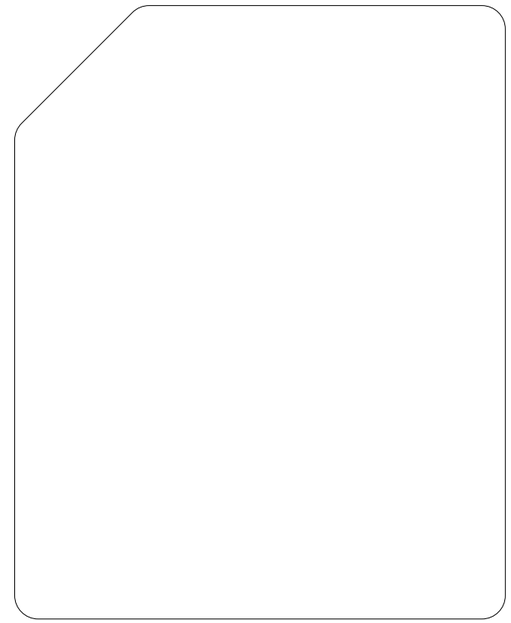
Das Darstellen von Informationen durch Verhalten kann emotionaler sein.

ZUNEIGUNG IST
DEINE RICHTUNG

EIN VEKTOR IN
///



01
DATENQUELLE



DAS MOBILTELEFON MEIN MOBILTELEFON IST

MEIN STÄNDIGER BEGLEITER. *Wie ein kleines Haustier muss ich es pflegen, Telefonbuch aufräumen & SMS löschen, ich muss es ernähren, durch aufladen des Akkus & ich kann mit ihm spielen. Mein Mobiltelefon ist ein kleines Haustier & es hat ein Gedächtnis das SIM-Karte genannt wird.*

DIE SIMKARTE TECHNISCH *Meine Datenquelle ist die SIM-Karte eines Mobiltelefons. Mit hilfe eines Kartenlesegeräts können SIM-Karten ausgelesen werden. Neben einigen persönlichen Daten wie etwa, die letzten 10 Anrufe, ein SMS-Archiv und bevorzugte Mobiltelefonnetze im Ausland, ist auf der SIM-Karte auch ein Telefonbuch gespeichert. Die Daten werden vom Telefon in Reihenfolge ihrer Eingabe auf der Karte gespeichert, dh wenn man nie eine Nummer aus seinem Telefonbuch gelöscht hat, repräsentiert die Reihenfolge der Einträge auch die zeitliche Abfolge wieder. Hat man jedoch bereits Einträge gelöscht werden diese Löcher beim nächsten neuen Eintrag wieder gefüllt und die chronologische Reihenfolge ist unterbrochen.*

Auf einer SIM-Karte können bis zu 150 Telefonnummer gespeichert werden.

Was genau auf den SIM-Karten gespeichert wird unterliegt anscheinend keiner Norm. Einige Mobiltelefone speichern ihre eigene Nummer auf der Karte ab, andere Archivieren SMS und manche speichern fast gar nichts, nicht einmal Telefonnummer. In den USA werden SIM-Karten in Mobiltelefonen nicht verwand.

INHALTLICH *Mein Telefonbuch ist ein Spiegel meines sozialen Umfelds. Meine Kontakte speichere ich in meinem Telefonbuch ab. In dem Moment in dem ich jemanden in mein Telefonbuch aufnehme, zeige ich, dass die Person eine Relevanz für mich hat. Die Nummern zeigen aber nicht nur mit wem ich mich beschäftige, sondern in einem Gewissen masse, durch die Vorwahlen der Festnetznummern, auch wo. Die SIM-Karte ist ein intimes Bauteil, um ihrer habhaft zu werden, muss man sein Mobiltelefon auseinanderbauen & und an seine Innereien. Das mag nicht jeder!*

SIM

SALES IN MINUTES
Simulation
SIMULATOR
SCANNING ION MICROSCOPE
SCREEN IMAGE MULTIMEDIA
Security Identity Module (cell phone usage)
SECURITY INFORMATION MANAGEMENT
SELECTED ION MONITORING
SELECTED ITEM MANAGEMENT
SEMANTIC INFORMATION MANAGER
SENSOR INPUT/INTERFACE MODULE
SERVICE INDUSTRY MARKETING
SERVICE INFORMATION MESSAGE
SERVIZIO INFORMAZIONE SEGRETO (ITALIAN SECRET SERVICE IN WWII)
SET INITIALIZATION MODE (HDLC)
SHORE INFRASTRUCTURE MODERNIZATION
SIGNAL INTERFACE MODULE
SINGAPORE INSTITUTE OF MANAGEMENT
SINGLE-ION MONITORING
SOCIEDAD DE INVERSIONES MOBILIARIAS
SOCIETY FOR INFORMATION MANAGEMENT
SOCIETY FOR THE INTERNET IN MEDICINE
SPARSE ITERATIVE METHOD
SPEAKEASY INFOSEC MODEL
SPECIAL INSTRUMENTED MISSILE
SPECIAL INTEREST MESSAGE
SPECTRAL INFORMATION MEASURE
STACK INTERFACE MODULE
STRATEGIC INFORMATION MANAGEMENT
STRESS-INDUCED MARTENSITE
SUBMARINE INTENDED MOVEMENT
SUBSCRIBER IDENTIFICATION MODULE (PCS)
SUBSCRIBER IDENTITY MODULE (ETSI GSM TECHNICAL SPECIFICATION)
SUDAN INTERIOR MISSION
SURAT IZIN MENGEMUDI (INDONESIAN DRIVING LICENSE)
SWITCH INPUT MATRIX
SYNCHRONIZATION INTERFACE MODULE (TELABS)
SYSTEM IMPACT MESSAGE
SYSTEM INTEGRATION MANAGER
SYSTEMS INTEGRATION MANAGEMENT
STYLE IN MOTION

Quelle
http://www.acronym-finder.com

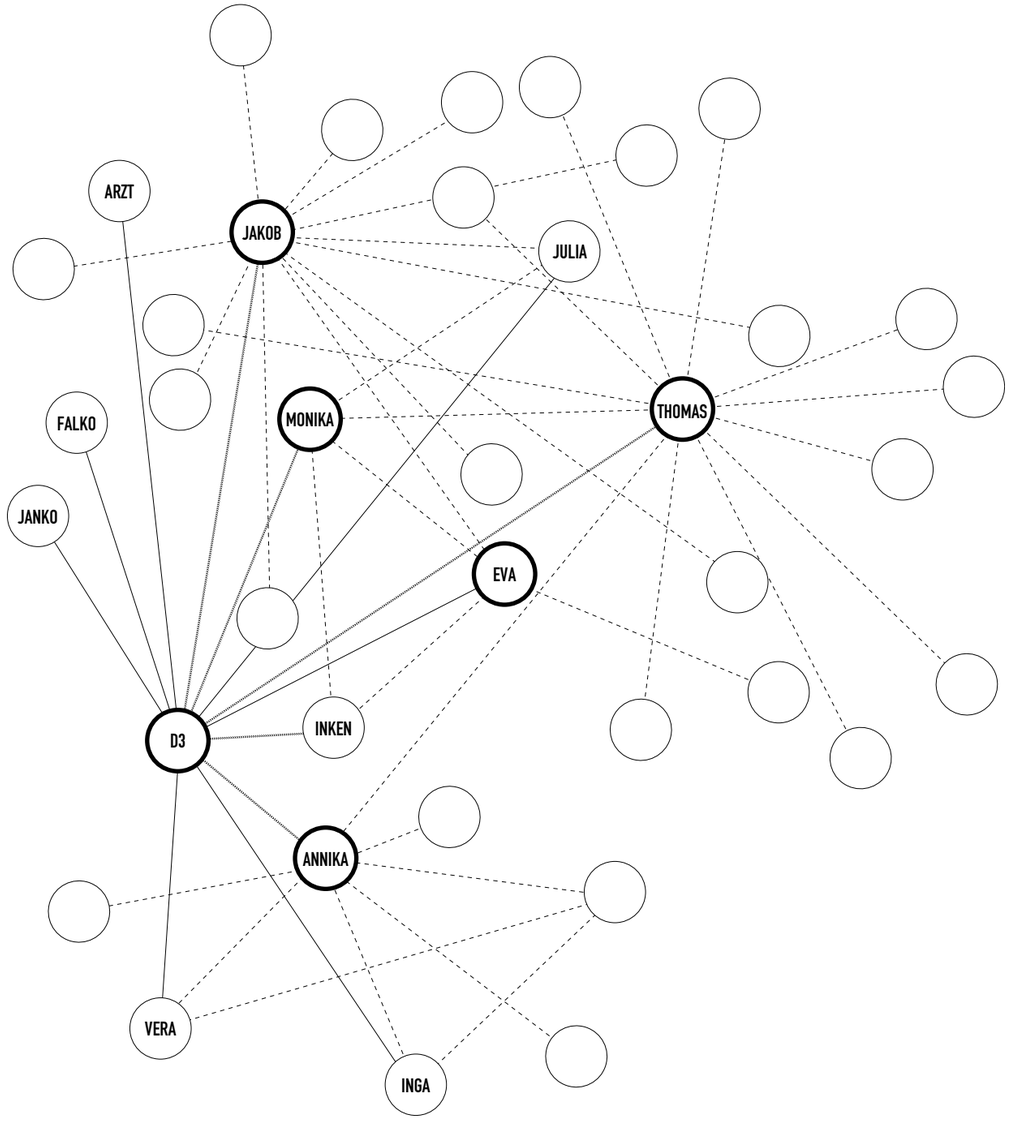
01793938833

1	ADAC PANNENHILFE	0179222222
2	ADAC VERKEHRINFO	017922411
3	ALEX	+495116068908
4	ANNELIE F	01706414915
5	ANNELIE FEST	+493029352529
6	ANNI HH	+494027881595
7	ANNI MOBIL	+491794508664
8	ANNI FRELSDORF	047498442
9	APPLY	05114850297
10	ARZT SCHRAMM	0304213606
11	AUSKUNFT	11881
12	BERND	+4528182247
13	CHRISTIAN	01794917882
14	ELLIE	01736104862
15	ESTHER	05113399905
16	MONIKA	03044324774
17	EVA MOBIL	01608207479
18	GEE FESTNETZ	04218969647
19	GEE VIAG	+491791076298
20	GFG	04213386810
21	GUNNAR	0421506580
22	HARRIET	0471413106
23	HEBAMMENPRAXIS	03041717321
24	HOMESWEETHOME	03042086652
25	INGA HOME	+494747872727
26	INGA MOBIL	01795306949
27	JAN KÖPER HDK	01791094705
28	JAN MOBIL	+491791066473
29	JANKO FEST	0511449647
30	JANKO MOB	01791254768
31	JHNDRICH	05121867179
32	JÖRG KÖNIG	+491723053400
33	JÖRN KRANKENBERG	+494748931560
34	KRISTIN	491704015792
35	KUNDENBETREUUNG	08005522255
36	MAIKE TD	01791125350
37	MAILBOX	+491793000333
38	MAMA ARBEIT	04213388112
39	MAMA BOKEL	04748931776
40	MAMA MOBIL	01794094576
41	MANNI	0307829126
42	MARC MOBIL	+491792006081
43	MARTIN LODOWN	01752090000
44	MATZE	04217949973
45	MATZE MAMA	04216360548
46	MATZE MOB	+491794610718
47	MEDIRENT	0308804100
48	MIRKO HDK	01736096593
49	MOLLI	0302177000
50	MONIKA MOBIL	01739912110
51	NICOLA MOBIL	+491777494397
52	NINA B	04791909002
53	NINA B MOBIL	+491793940098
54	NINA+G	03027574865
55	NONNO	0421592675
56	PROJEKTIONISTEN1	03044717500
57	HOLGER EP	01637354758
58	OLLI	+495113180525
59	OLLI MOBIL	01792975189
60	PROJEKTIONISTEN	+495111317496
61	P_BJÖRN MOB	+491772534360
62	P_HENRIK	+49511672630
63	P_HENRIK MOB	+491776252489
64	P_MARTIN MOB	+491795922828
65	P_MATTHIAS MOB	+491772534370
66	P_TOBIAS MOB	+491773421416
67	P_BJÖRN FEST	+49511802771
68	ROBBEN UND WIENS	030421036
69	SEB MOBIL	+491738701313
70	SUSANNE	01739937725
71	SIMONE BOKEL	+494748821268
72	STEPHAN K MOB	+491795150213
73	STEFANO	0041786241869
74	TAREK NEU	01637494007
75	TAXI B	03069022
76	TAXI HH	040666666
77	TELEKOM WECKEN	01801141033
78	THOMAS TIETJEN	+491776465673
79	TIM HB	04215977397
80	TIM MOB	+491792401743
81	FALKO ARBEIT	03042081605
82	TIMO HDK MOBIL	01721688268
83	UBBEN	01728856264
84	VERA MOBIL	+491793908504
85	VOLKER MOBIL	01772738620
86	WT B	0306035787
87	WIEBKE WERNER	01724193128
88	ZEITANSAGE	0179934836
89	JENS HESSE	01732077912
90	JAKOB LEHR	01777451233
91	ANDREAS WT	03042010051
92	ANNE HDK	01742410646
93	ANNE HDK FEST	03082309038
94	HANNES HDK	01792437826
95	EF	+494039803581
96	CHRISTIAN BORM	+491747050063
97	TIMO GAESSNER	01723950069
98	STEFANO.CH	0041786241869
99	SIGO WT	01743653985
100	ANNI ARBEIT	03082722350
101	PETER SI HING	01792507016
102	KH JERON	01702127673
103	ROMAN WT	03025296134
104	ANDREAS WT WORK	03038651573
105	KPMG SPANIER	03020681535
106	BODIL	01793939274
107	DEUTSCHE BAHN	01805996633
108	OLE	01796758136
109	RESTAURANT SUD	03042801846
110	THOMAS T HOME	03044050345
111	NINA+G MOB	01748145380
112	STEPHAN KÖSTER	04040197945
113	PETER W	01795170110
114	JULIA GUTHER M!	01796766284

01793938833
die Telefonnummer meines Mobiltelefons.
Zu sehen ist der komplette Inhalt meines Mobiltelefonbuchs.
Stand 20 12 2002

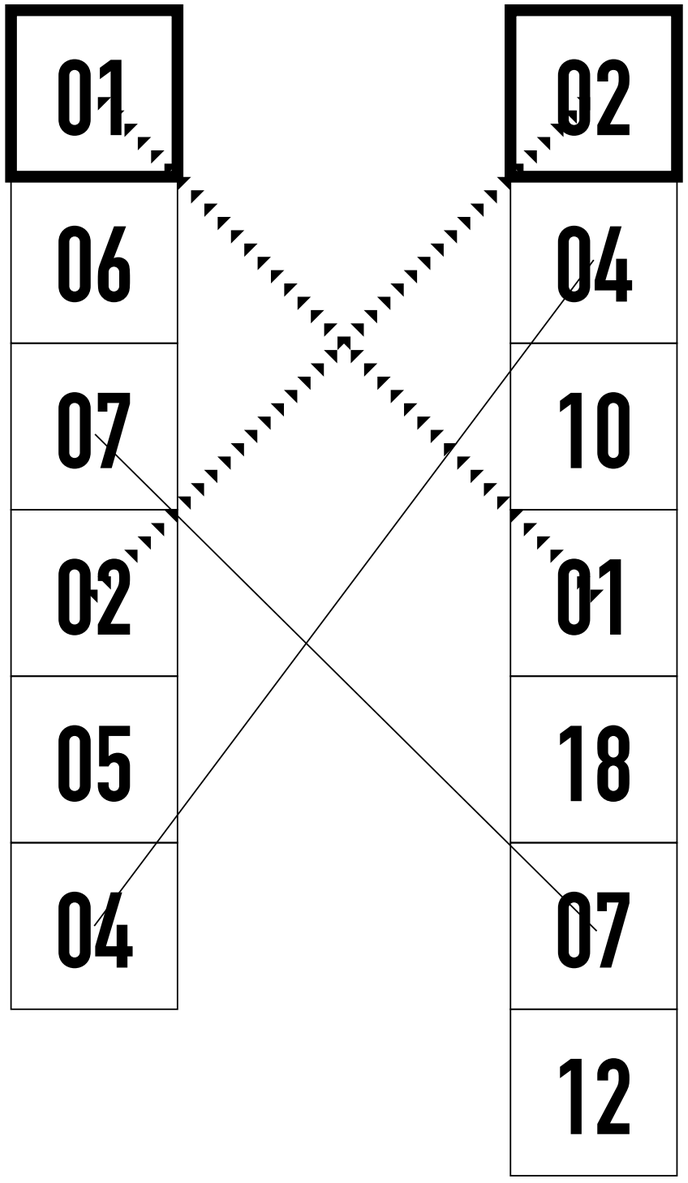
SOZIALES NETZ

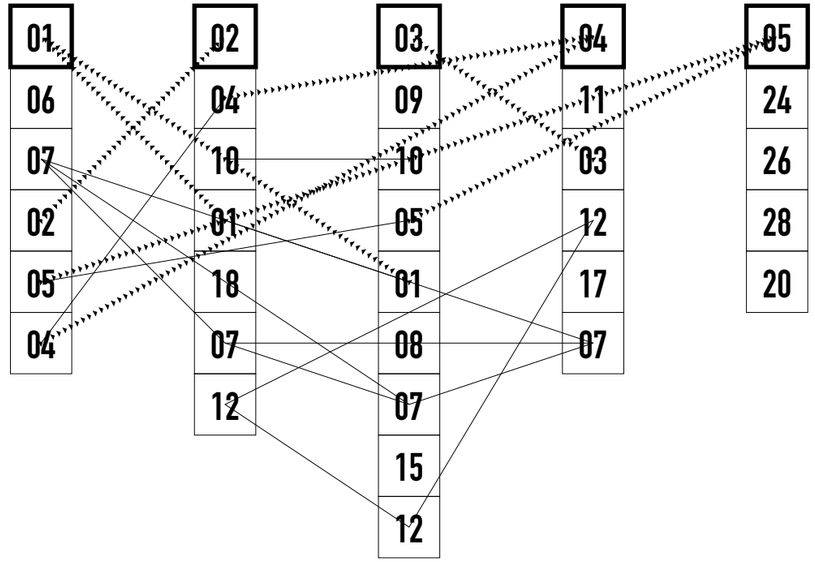
Betrachtet man nicht nur eine Karte sondern kombiniert man die Daten von mehreren SIM-Karten, so können Netze aus sozialen Verbindungen sichtbar werden. In der Zeichnung ist beispielhaft ein Ausschnitt aus meinem Bekanntenkreis aufgezeigt.



MÖGLICHKEITEN DER DATEN

Es gibt ein ganze Menge Wege die Daten aus verschiednen Telefonbüchern miteinander zu Kombinieren. Die Frage sich stellt ist, welche dieser möglichen Daten auch relevant sind. In der Zeichnung sind eine Reihe Schlüsse aufgezeigt, die zum grossen Teil auch in der eigentlichen Arbeit benutzt werden. Folgende Möglichkeiten werden hier gezeigt ->
A *ich kenne dich* **B** *wir kennen uns* **C** *wir haben x gemeinsame Bekannte* **D** *ich habe mehr/weniger Bekannte als du*

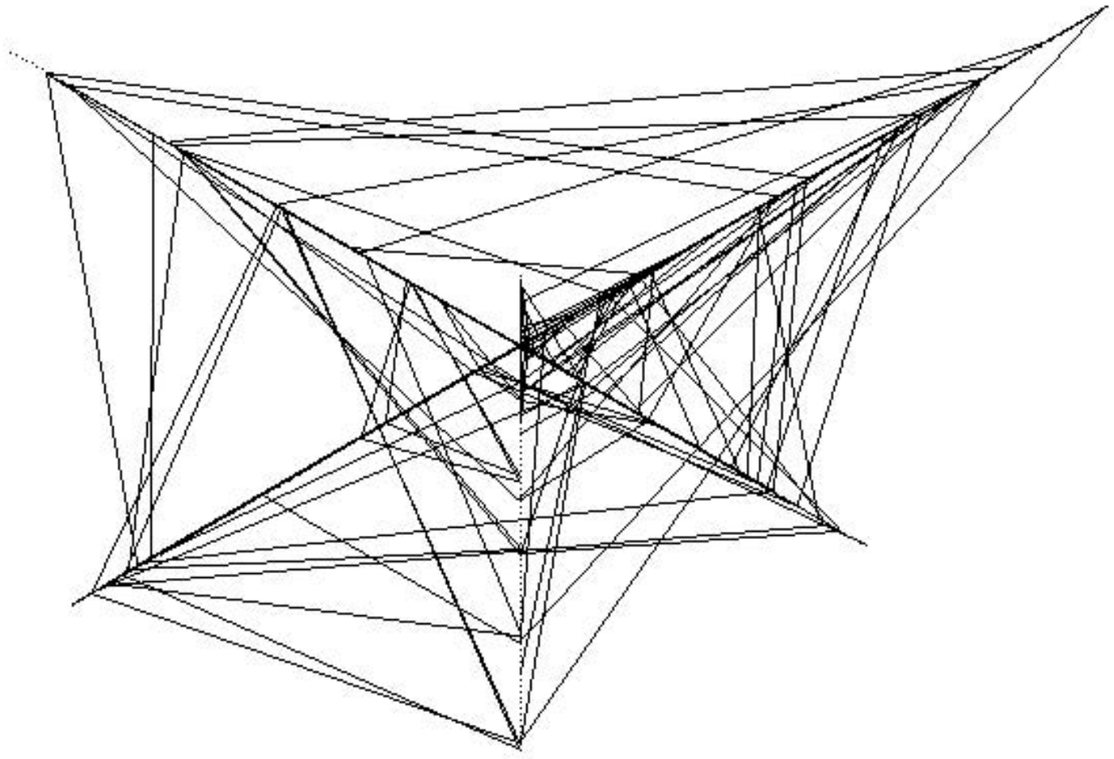


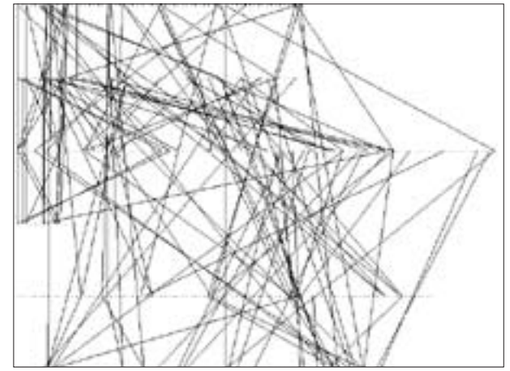


GESAMTZAHL EINTRÄGE	5	6	8	5	4
DAVON AUCH ANDEREN BEKANNT	4 = 07, 02, 05, 04	5 = 04, 10, 01, 07, 12	5 = 10, 05, 01, 07, 12	3 = 03, 12, 07	/
DAVON ANWESEND	3 = 02, 05, 04	2 = 04, 01	2 = 05, 01	1 = 03	/
DAVON GEGENSEITIG BEKANNT	1 = 02	1 = 01	/	/	/
MICH KENNEN	2 = 02, 03	1 = 01	1 = 04	2 = 01, 02	2 = 01, 03

GEMEINSAME BEKANNTE					
	(ME)	04, 07	05, 07	07	/	mit 01//
	07, 04	(ME)	10, 01, 07, 12	07, 12	/	mit 02//
	07, 05	10, 01, 07, 12	(ME)	07, 12	/	mit 03//
	07	07, 12	07, 12	(ME)	/	mit 04//
	/	/	/	/	(ME)	mit 05//

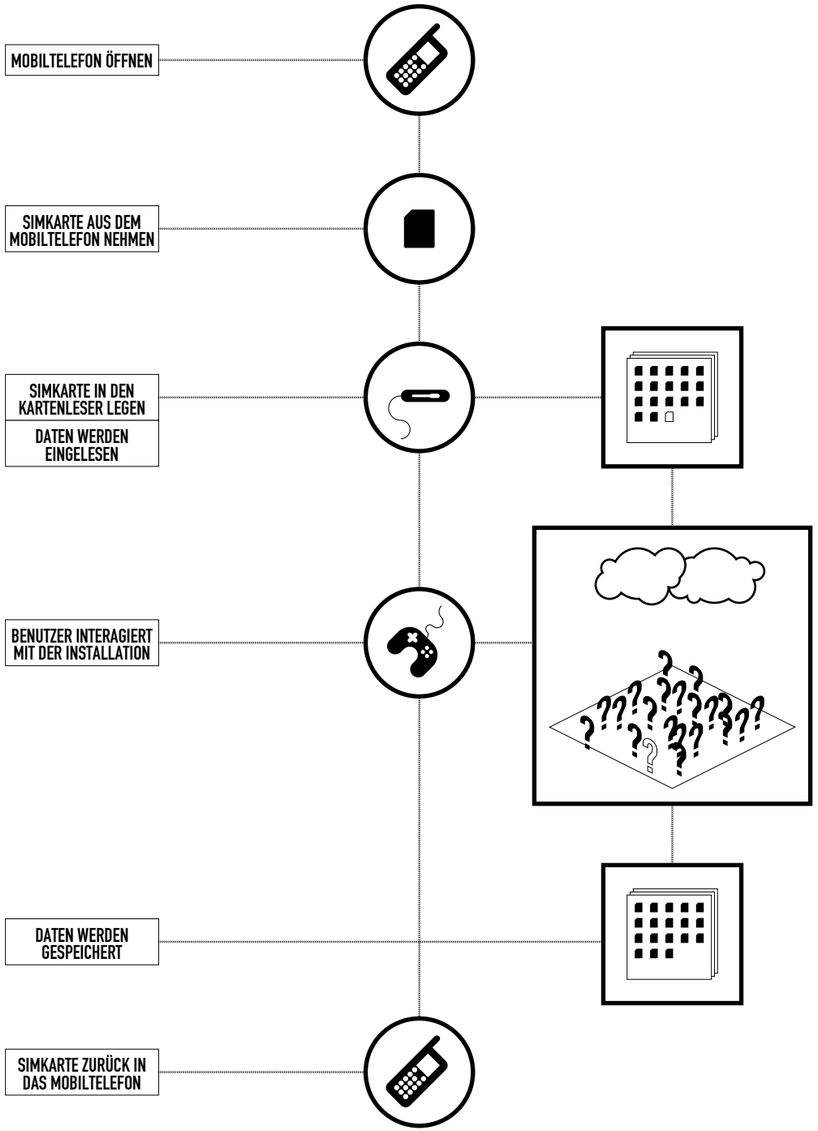
›››*Diese Skizze wurde mit einem Tool erstellt, dass ich geschrieben habe um die Verknüpfung an ›echten‹ Daten zu untersuchen. Die Daten sind einmal auf einem radialen Koordinatensystem abgebildet und in dem oberen Bild rechts in einem Rechtwinkligen. Es werden 6 Datensätze dargestellt, die zuvor mit einem Lesegerät erfasst wurden. Ausgehend vom Zentrum, werden alle Einträge eines Telefonbuchs als Punkt auf einer Linie dargestellt. Alle gleichen Nummern werden mit einem Strich verdunden und ergeben so die spinnennetzartige Struktur. Die Abbildung ist bei nur 6 Teilnehmern schon sehr komplex.*





02
ABLAUF

& ICH HOLE DIE SIMKARTE AUS MEINEM MOBILTELEFON
EIN INIMTER MOMENT //
///////////////////////////////



ABLAUF

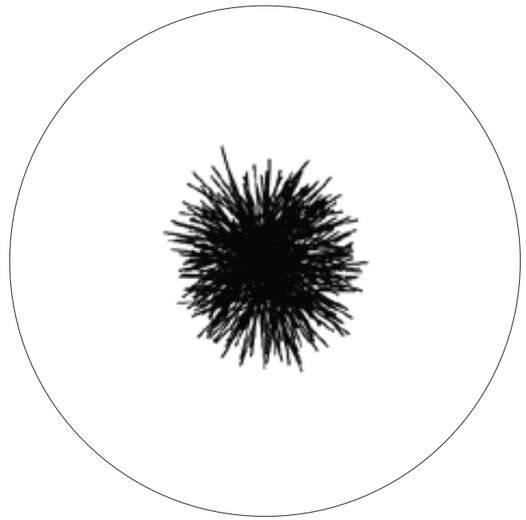
Meine Arbeit ist eine lokale Installation. Vor Ort werden, mit einem SIM-Kartenlesegerät, das Telefonbuch aus einem Mobiltelefon, in der Installation dargestellt & gespeichert. Die Daten werden gesammelt, so dass die Anzahl der gespeicherten Telefonbücher nach & nach wächst. Die Schnittstelle mit der der Benutzer mit der Installation in Kontakt tritt, ist durch ein Joypad realisiert. Dargestellt wird die Visualisierung durch eine Beamer-Projektion auf einen Tisch. Die Installation teilt sich grundsätzlich in drei Schritte.

01 DAS EINLESEN DER DATEN *Ein Benutzer tritt an die Installation, holt die SIM-Karte aus seinem Mobiltelefon, legt sie in das Lesegerät. Die Daten werden geladen und die Interaktion mit der Installation beginnt.*

02 DAS ERFORSCHEN *Der Benutzer kann mit dem Joypad eine Herde aus seinen Telefonnummern über eine 3Dimensionale Landschaft steuern. Die Herde setzt sich aus den einzelnen Telefonbucheintrag zusammen. Der Benutzer kann jetzt mit anderen Objekten in Kontakt treten und durch deren Verhalten ablesen, wie der Kontakt zu anderen Objekten geartet ist. So kann der Benutzer von einem Objekt zum anderen fahren, wenn er alle angekuckt hat oder keine Lust mehr hat, beendet er den Modus & sein Objekt wird in die Welt aufgenommen. Sie wird ein Teil der Simulation.*

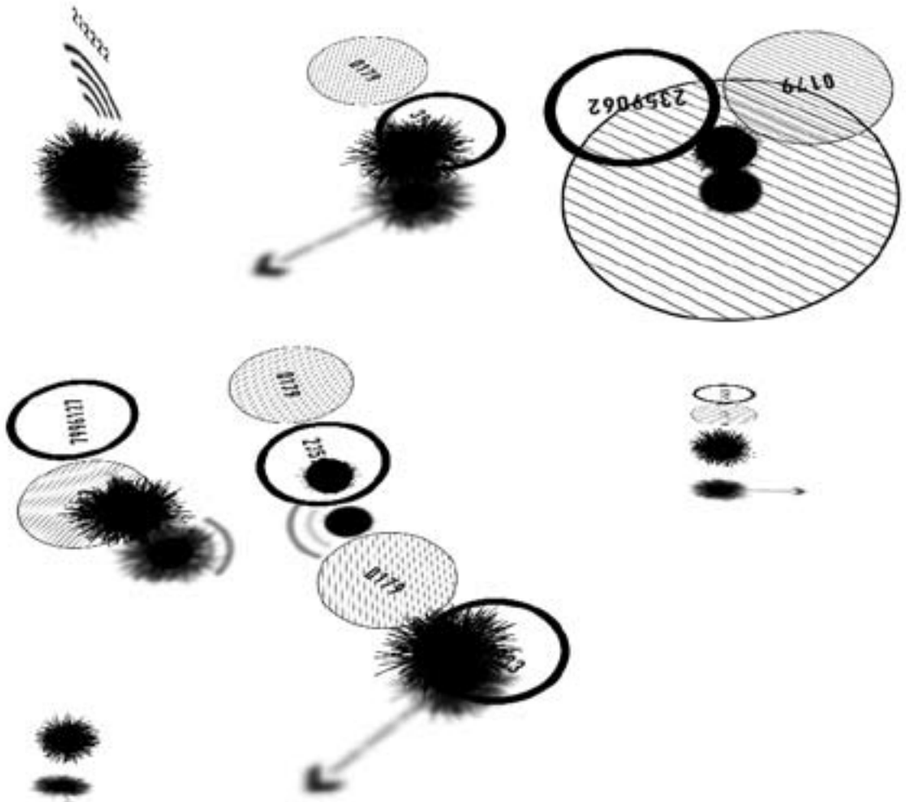
03 DIE SIMULATION *In der Simulation kann der Benutzer mit dem Joypad eine Kamera durch die Welt steuern und kucken, was passiert. Die anwesenden Objekte haben je nach Art ihrer Beziehung, ein bestimmtes Repertoire an Verhalten; Kennt ein Objekt ein anderes ›persönlich‹ dh es hat die Telefonnummer im Telefonbucher, besteht eine Freundschaft und diese beiden Objekte bilden eine Gruppe & artikulieren ihre Freude. Kennen die beiden Objekte die gleichen Personen dh sie haben gleiche nummern in ihrem Telefonbuch, dann ist es wahrscheinlich, dass sie beieinander bleiben. Je grösser die Anzahl der Übereinstummungen, desto grösser die Wahrscheinlichkeit, dass sie beieinander bleiben. Gibt es keine Übereinstimmungen, verjagt ein Objekt das andere. Wer wen verjagt hängt von seiner sozialen Stärke ab. Soziale Stärke ist die Anzahl der Kontakte zu anderen Objekten. Die Objekte finden sich also almählich zu Gruppen zusammen, die sich allerdings ständig neu ordnen und beim eintreffen eines neuen Objektes auch durchaus anders zusammensetzen können.*

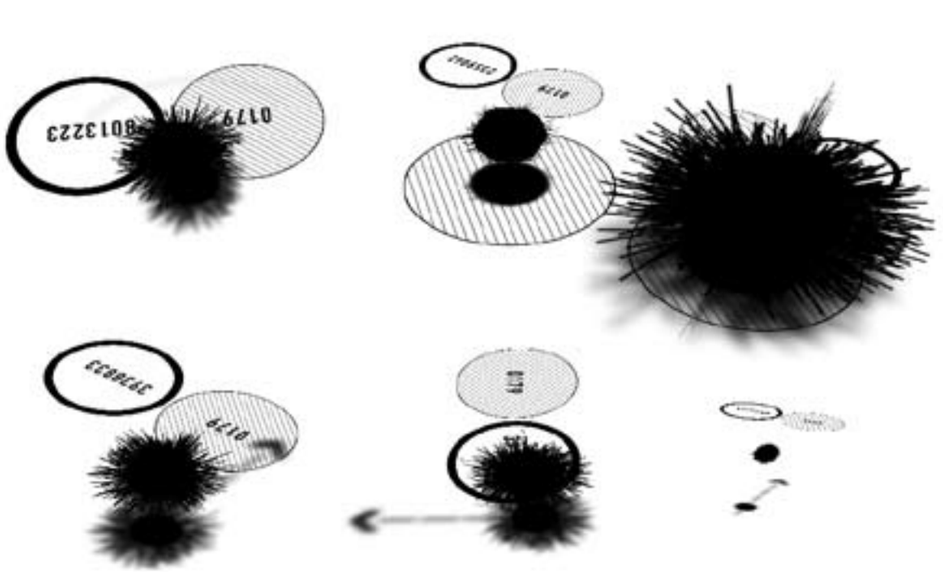
03
ERSCHEINUNG



FORM DER OBJEKTE

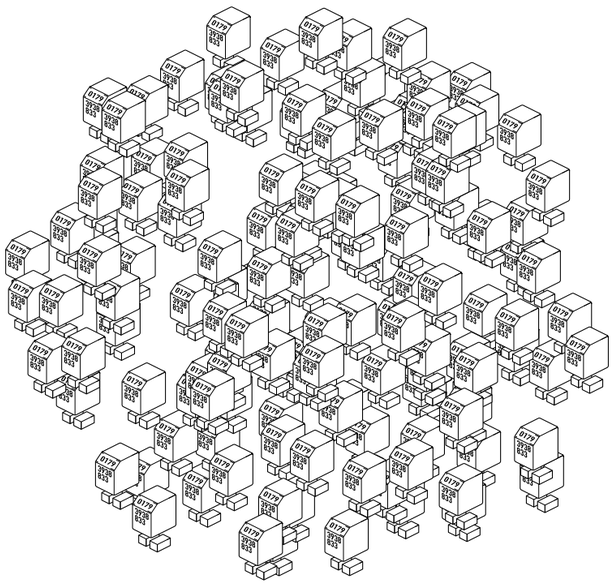
Die Form der Objekte, der ›Simthings‹, ist zurückhaltend und unterstütz nur die Darstellung von Verhalten. Im Prinzip sind die vier rechts gezeigten Formen die Elemente aus denen die Objekte zusammengesetzt sind. Unten sind diverse Zustände eines Objektes gezeigt.
Im Anhang befinden sich unter ›Skizzen‹ entwürfe anderer Formen.

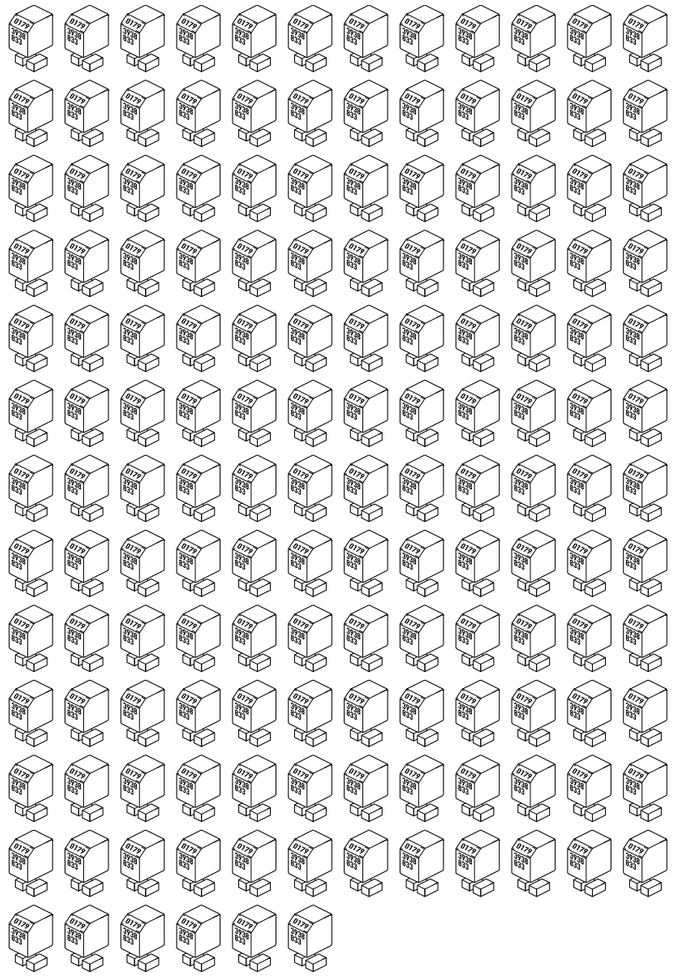




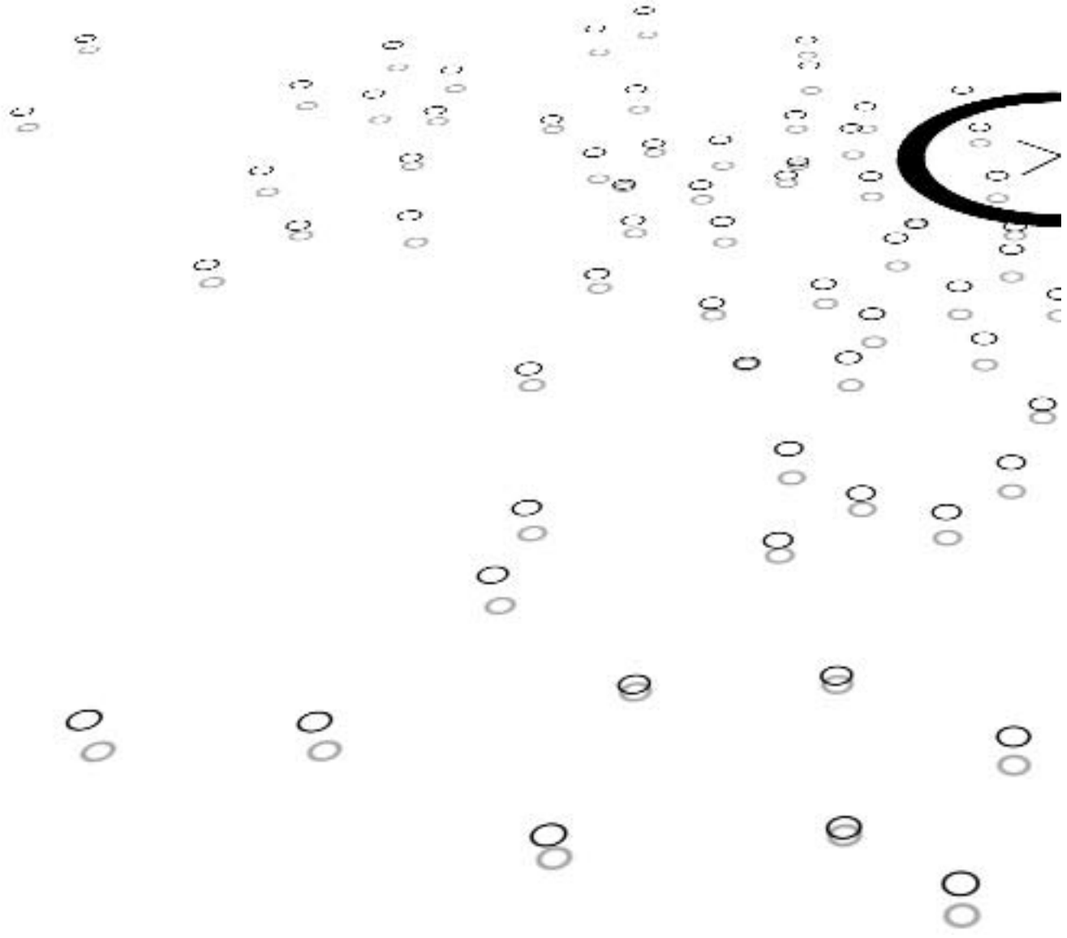
WIEVIEL IST 150

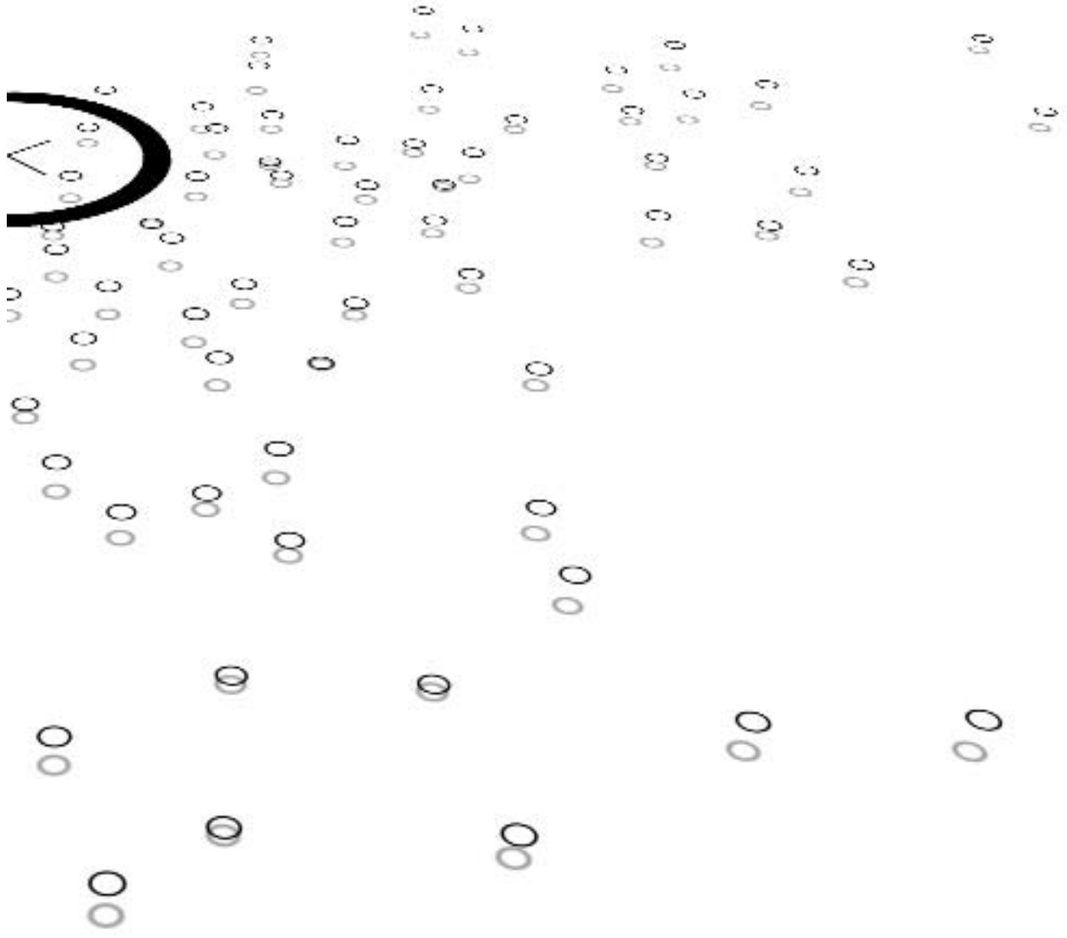
Eine SIM Karte kann maximal 150 Einträge haben, deshalb 150. Das bedeutet sowohl technisch als auch gestalterisch; ein grosse Menge von Objekten. Um die Zahl begreifbar zu machen sind hier einmal 150 Objekte geordnet und einmal 150 Objekte in ihrer Positionierung ungeordnet.





DIE HERDE *Nachdem die Daten des Benutzers eingelesen wurden, wird aus ihnen, bevor sie zu einem Objekt werden zunächst eine Herde von Telefonnummern generiert. Jeder Ring repräsentiert eine Telefonnummer.*





GUI GESTALTUNG

Die Gestaltung des Graphical User Interface betsteht aus zwei Teilen. Einerseits aus den Nachrichten, die dem Benutzer helfen sich durch die Anwendung zu finden und andererseits aus der Darstellung der momentanen Belegung der Tasten des Joypads.

›On Screen Instructions‹

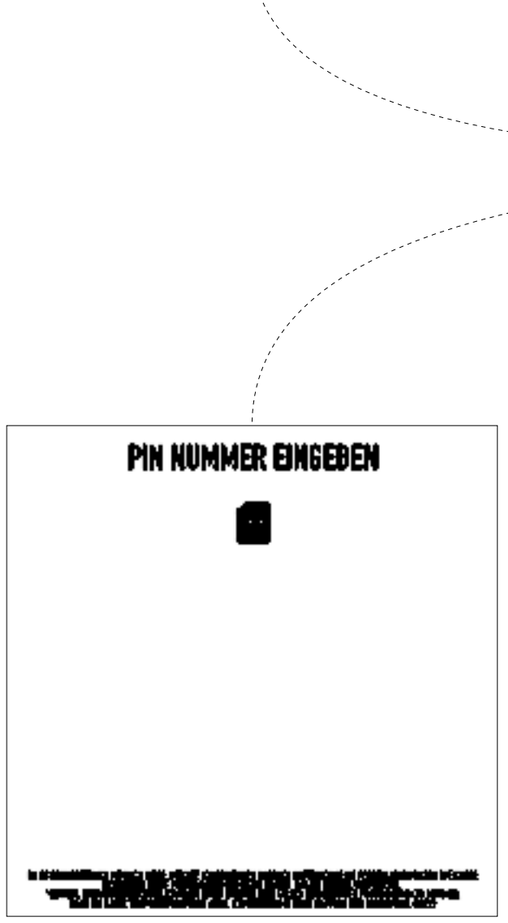
›On Screen‹ Erklärung der Tastenbelegung

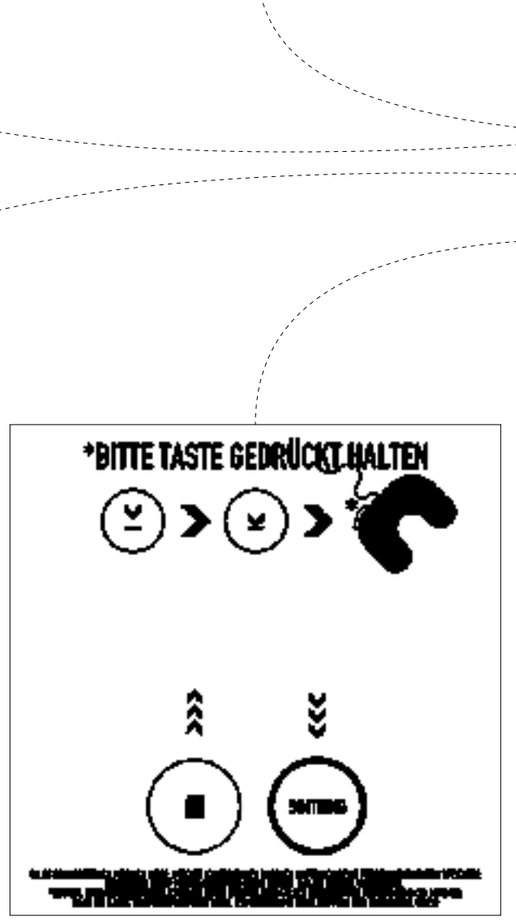

SIM KARTE AUS DEM TELEFON NEHMEN
JOYPAD UMDREHEN UND KARTE EINLEGEN

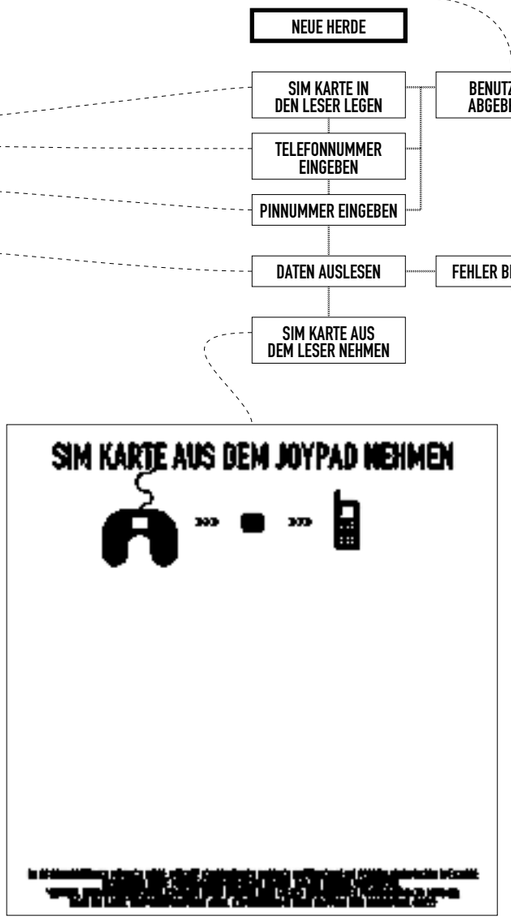

TELEFONNUMMER EINGEBEN


HMMMM, SELBER ABGEBROCHEN, JA?


AUS DEN DATEN WIRD EIN SIMTHING ERZEUGT


PIN NUMMER EINGEBEN

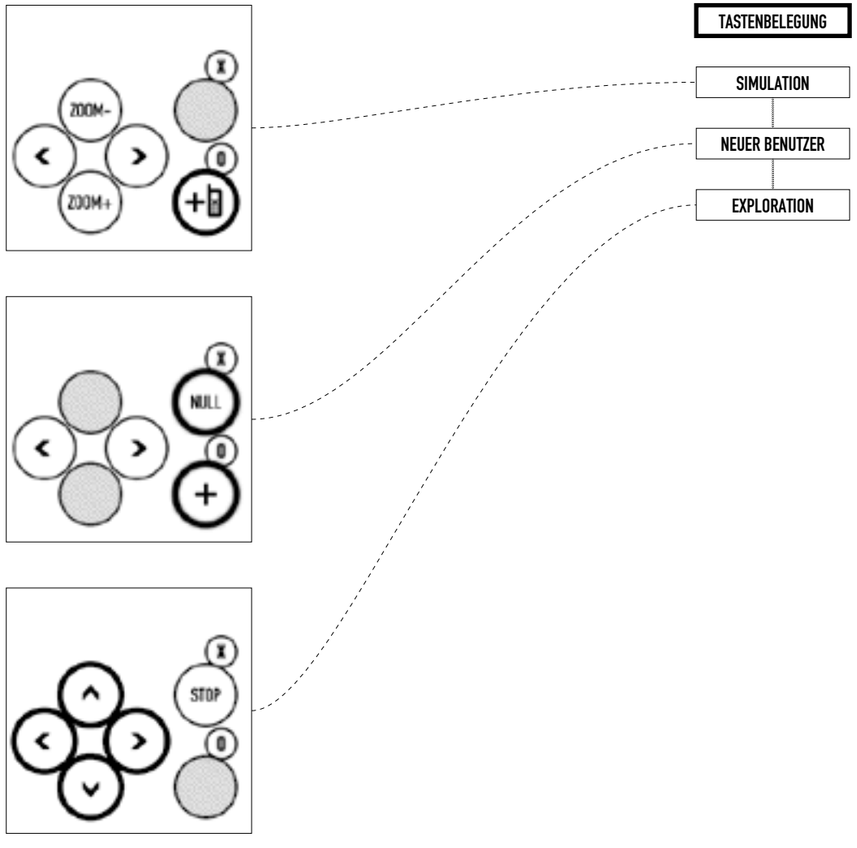

*BITTE TASTE GEDRÜCKT HALTEN


SIM KARTE AUS DEM JOYPAD NEHMEN


FEHLER BEIM LESEN DER SIM KARTE

NEUE HERDE
NEUES SIMTHING
SIM KARTE IN DEN LESER LEGEN
BENUTZER HAT ABGEBROCHEN
AUS DEN DATEN WIRD EIN SIMTHING ERZEUGT
TELEFONNUMMER EINGEBEN
PINNUMMER EINGEBEN
DATEN AUSLESEN
FEHLER BEIM LESEN
SIM KARTE AUS DEM LESER NEHMEN

>>> TASTENBELEGUNG *In der rechten unteren Ecke wird die Tastenbelegung des Joypads mit hilfe von Symbolen dargestellt. Das Joypad hat ein Steuerkreuz mit den Richtungen oben, unten, links und rechts, sowie zwei Knöpfe. Die Knöpfe werden mit einer postiven und einer negativen Funktion belegt. ›O‹ ist positiv und wird für weiterführenden Aktion benutzt, ›X‹ ist negativ und wird zum Abbrechen und Zurücksetzen benutzt.*
SIMULATION *Im Simulationsmodus beschränkt sich die aktivität des Benutzers auf das Beobachten, deshalb kann mit links und rechts der Fokus auf ein anderes Simthing gewechselt werden, mit oben und unten wird der Kamerazoom verändert und durch drücken der ›O‹ Taste wird in den ›Neuer Benutzer‹ Modus gewechselt.*
NEUER BENUTZER *In diesem Modus müssen sowohl Telefon- als auch PINnummer eingegeben werden. Mit links und rechts wird der Eingabecursor bewegt mit ›O‹ wird die fokusierte Zahl hochgezählt und mit ›X‹ wird sie wieder auf Null gesetzt. Die Steuerung ist in dem Moment in dem die PINnummer eingegeben werden muss in einer schwierigen Situation, da die Installation öffentlich ist, die PINnummer aber privat. Aus diesem Grund macht man sich mit der Eingabemechanik beim eingeben der Telefonnummer vertraut und gibt die PINnummer dann blind ein.*
EXPLORATION *Im Explorationmodus, dienen die Richtungen zum steuern der eigenen Herde. Mit der ›X‹ wird die Exploration verlassen.*



Das Joypad; auf seiner Rückseite ist ein Kartenleser angebracht.

04
VERHALTEN

EINE MÖGLICHE DEFINITION VON VERHALTEN

In meiner Arbeit definiere ich Verhalten als die Kombination von Bewegung einer Form, in einem Kontext; dh eine Form stellt ohne Bewegung kein Verhalten dar, ausser natürlich in genau der Abwesenheit von Bewegung, was aber auch in die Kategorie der Bewegung fällt. Eine Bewegung ohne Form ist nicht sichtbar und der Kontext lenkt, die beobachete Bewegung in assoziative Bahnen. Es gibt noch viele andere Defintionten von Verhalten, so wird in der Biologie zB zwischen Verhalten das durch ein Lebewesen selbst ausgelöst wird und eins das durch die Umwelt ausgelöst wird, unterschieden. Verhalten hängt sehr oft mit Personifizierung zusammen. Eine Beobachtung die mich sehr oft beeinflusst hat beim Gestalten meiner Arbeit ist die tatsache, dass Leute schon bei simpelsten Formen in Bewegung anfangen, diesen Eigenschaften von Lebewesen zuzuschreiben. So ist der Strich der ständig vom Bildschirmrand abprallt, es aber immer wieder versucht, dumm, oder der Pfeil der ständig auf und abhüpft, lustig. Oft bilden sich im Kopf des Betrachters, in dem Moment in dem eine Interaktion zwischen verschieden Formen stattfindet, Charaktere. So ist der Pixel der einen nach dem andern Pixel über den Bildschirm verfolgt und schliesslich ›auffrisst‹, irgendwann soetwas wie das Raubtier und die anderen sind Opfer.

IMPLEMENTIERTE VERHALTEN

im folgenden werden alle Verhalten beschrieben, die Beobachtet werden konnten.
Interne Verhalten werden nicht durch äussere Reize ausgelöst. Exteren Verhalten werden durch Veränderungen der Umgebung hervorgerufen.
Die Internen und die Externen sind bewusst programmiert, die Emergenten entstehen aus der Komibination der ersten beiden. Deshalb ist es durchaus möglich, dass noch weitere nicht beschriebene Verhalten zu beobachten sind.

INTERNE

Wandern
Schlafen

EXTERNE

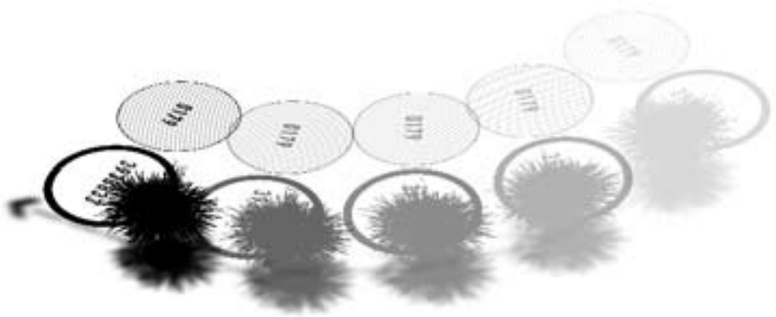
Sympathie
Freude
Rivalität
Angeben
Fliehen

EMERGENTE

Gruppen bilden

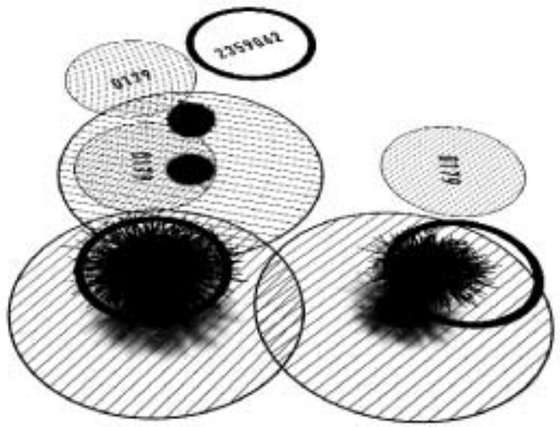
>>> INTERNE

WANDERN *Wandern ist das ›standard‹ Verhalten. Die meiste Zeit werden die ›Simthings‹ dieses Verhalten zeigen. Wie viele anderen Verhalten, hat auch ›wandern‹ Unterverhalten. Ein wesentliches ist die Neigung in einem bestimmtes Gebiet nicht zu verlassen. Diese Eigenschaft führt dazu, dass die ›Simthings‹ in der Nähe der Anderen bleiben. Das eigentliche wandern ist eine völlig zufällige Bewegung.*

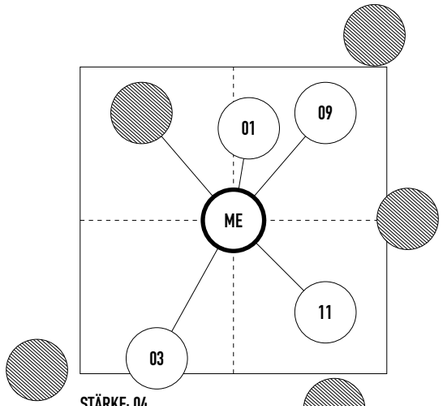


SCHLAFEN *Dieses Verhalten ist nicht wirklich sinnvoll. Von Zeit zu Zeit fallen die ›Simthings‹ in einen Schlaf, in dem sie von Nummern aus ihrem eigenen Telefonbuch träumen.*

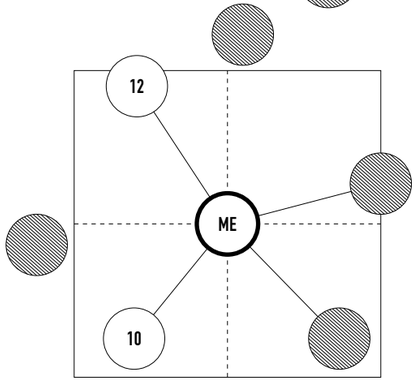
SYMPATHIE *Dieses Verhalten bewirkt, dass sich die Objekte nicht mehr so schnell bewegen, also in der Nähe voneinander bleiben.*



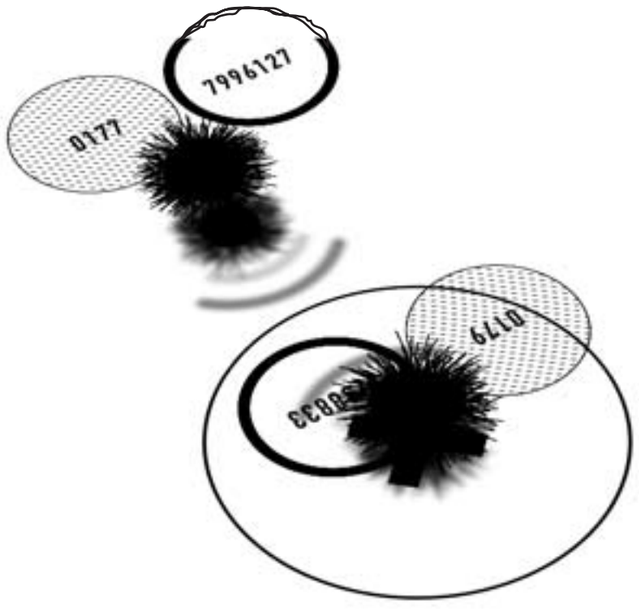
RIVALITÄT *Dieses Verhalten ist ein Schlüssel-Verhalten, da es dazu führt, da es wesentlichen anteil daran hat, dass sich die ›Simthings‹ räumlich, nach Bekanntschaften sortieren. Wenn sich zwei Objekte treffen und diese sich nicht kennen, dh sie weder die Telefonnummer des anderen, noch irgendeine ander Bekannstschaft haben, kommt es zu dem Rivalitätsverhalten. Die beiden Objekte zappeln für eine kurze Zeit voreinander herum und skannen sich gegenseitig. Danach gibt es entweder ein Unentschieden oder einen Gewinner und einen Verlierer. Wer Verlierer und wer Gewinner ist, errechnet sich aus der momentanen Stärke der Objekte. Je mehr bekannte Objekte in der Nähe eines Objektes sind, desto grösser ist seine Stärke.*



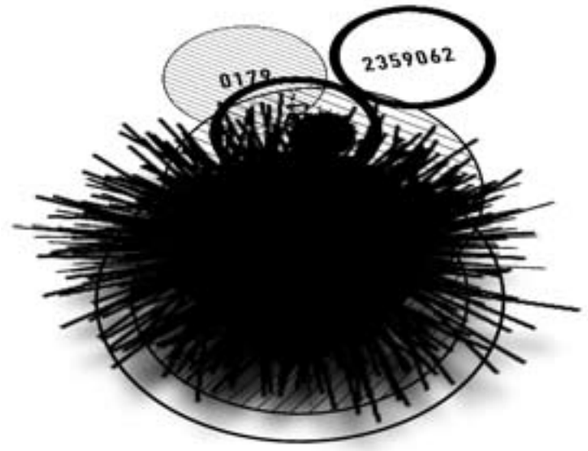
STÄRKE: 04



STÄRKE: 02



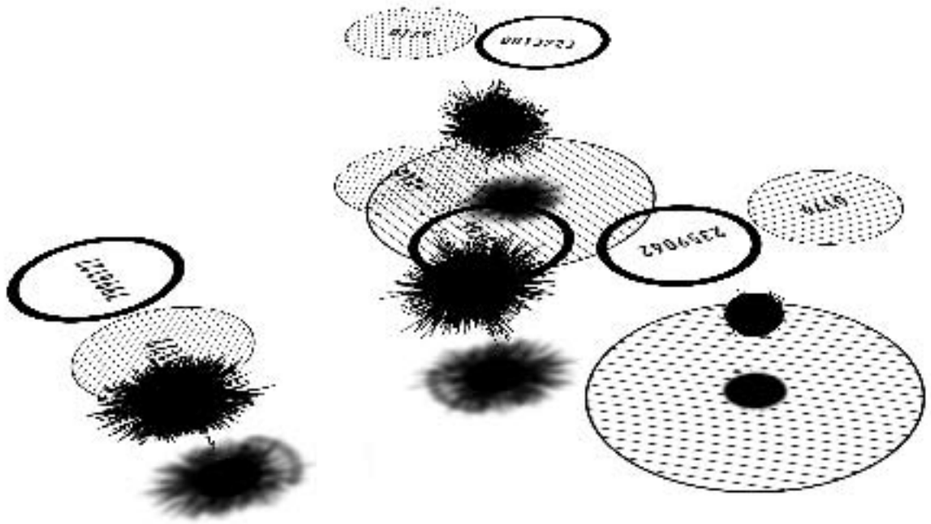
GEWINNER *Das Resultat einer Rivalität kann einen Gewinner und einen Verlierer produzieren. Der Gewinner bleibt an seinem Ort und bläht sich auf.*



VERLIERER *Der Verlierer flüchtet in eine andere Richtung. Die Kombination dieser Verhalten führt dazu, dass sich nach und nach die Objekte sortieren, da Freunde immer beineinander bleiben und Unbekannte fliehen.*

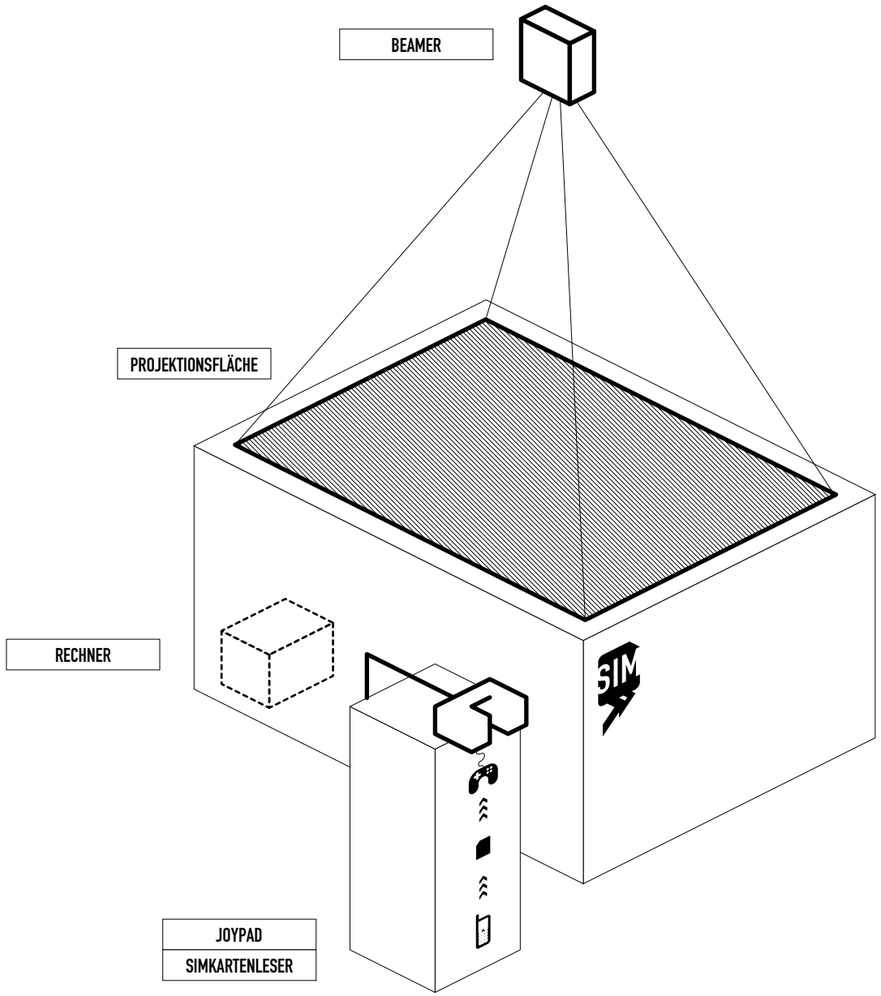
>>> EMERGENT

GRUPPEN BILDEN *Als solches nicht implementiert resultiert das bilden von Gruppen aus der kombination von Wandern und Rivalität. Je komplexer Verhalten miteinander kombiniert werden, desto mehr emergente Verhalten können entstehen.*

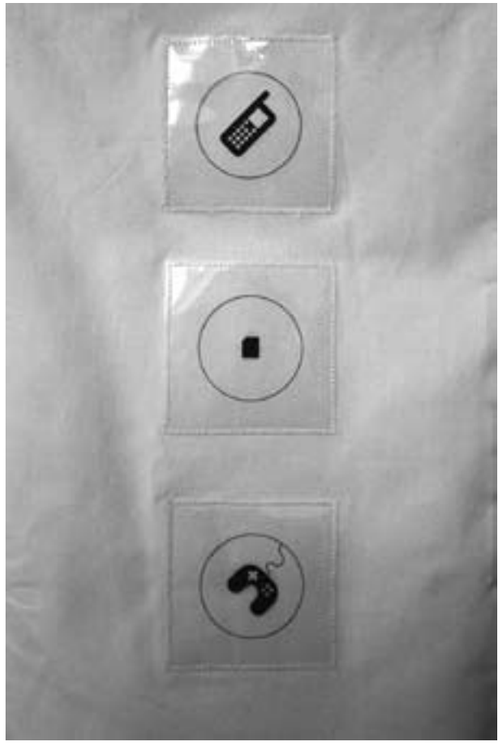


05
AUFBAU

›››*Mit hilfe eines Beamers wird auf einen 120cm mal 90cm grossen Tisch projeziert. Unter dem Tisch steht, nicht sichtbar, ein kleiner Rechner, an den der Kartenleser und das Joypad angeschlossen sind. Die beiden Eingabegeräte liegen auf einem 30cm mal 30cm grossen Tisch. Das Kartenlesegerät ist an der Unterseite des Joypad angebracht.*



JOYPAD UND KARTENLESER *Der hier sichtbare Kartenleser ist eigentlich nur eine Halterung für SIM Karten, entliehen aus einem alten Nokia Mobiltelefon. Die tatsächliche Kartenlesertechnik ist nicht sichtbar.*



MATERIAL *über die beiden Tische wird jeweils eine Stoffhusse gestülpt. Auf den Stoff wurden zur Applikation passende Grafiken genäht. Stoff ist ein ideales Material für Projektionen.*

ZOOM+
ZOOM-

///

A01

TECHNOLOGIE

& JAVA1.4 / LWJGL0.5 / GL4JAVA / OPENGL / OCF / WINDOWS XP / HMMMMM...

SOFTWARE

Um eine gute Kombination aus Portabilität, Geschwindigkeit und Komplexität zu erhalten, hab ich mich dazu entschlossen, meine Arbeit in **JAVA** *zu programmieren. Das stellt ein besondere Herausforderung an mich, da man als Gestalter für gewöhnlich Tools wie Macromedia Director oder Macromedia Flash benutzt, die sich durch ihre einfache Integration von verschieden Medien, wie Sound, Film, Vektor- & Pixelgraphik auszeichnen.*
Der Vorteil von Java ist ganz sicher die höhere Geschwindigkeit und die grosse Unterstützung in vielen im Netz zugänglichen Projekten. So gibt es für Java ein Projekt von IBM mit dem Namen **OPEN CARD FRAMEWORK (OCF)**, *dass die integration von Kartenlesegeräten in Java Anwendungen erlaubt.*
Für die Ausgabe der 3DGraphik habe ich mich für **OPENGL** *entschieden. Für Java gibt es zwei verschiedene Technologien OpenGL zu benutzen. Entweder das OpenGL-Binding mit dem Namen* **GL4JAVA**, *welches es ermöglicht OpenGL Befehl in Java auszuführen oder ein OpenSource Projekt mir dem Namen* **LEIGHT WEIGHT JAVA GAMING LIBRARY (LWJGL)**, *dass eine Reihe von Klassen zur Verfügung stellt, die zum entwickeln von Spielen nötig sind. Grafik (OpenGL), Sound (OpenAL) & Eingabegeräte (Joystick, Joypad). Im Prinzip ist* **LWJGL** *für Windows, Linux und Mac OS X verfügbar, da allerdings der Port für OS X zur Fertigstellung meines Diploms noch nicht verfügbar war, musste ich auf einen Windows-Rechner wechseln; Windows XP also, hmmm…*

HARDWARE

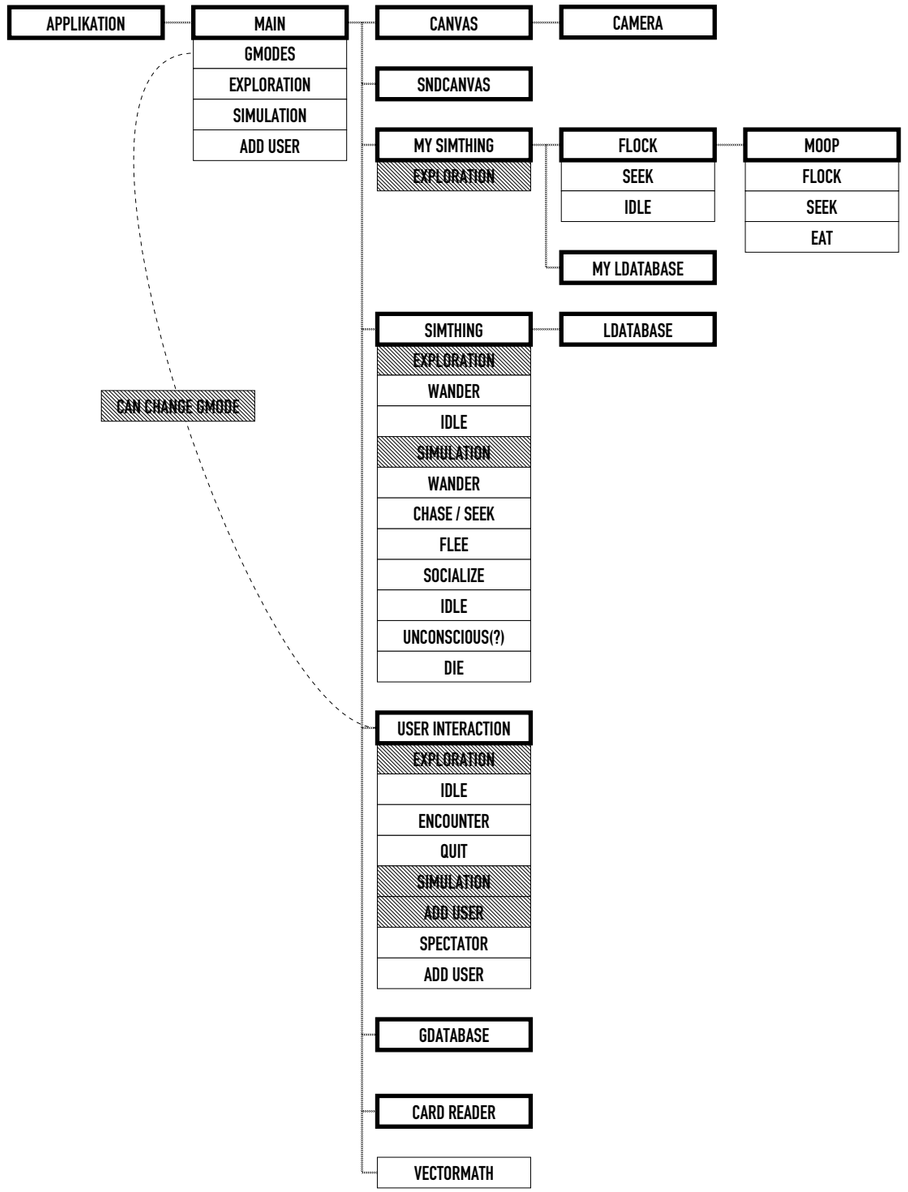
Zum realiseren der Installation wurden folgende Geräte benötigt; Ein **BEAMER** *für die Projektion der graphischen Ausgabe auf einen Tisch, ein* **JOYPAD**, *durch das der Benutzer mit der Applikation kommunizieren kann, ohne dass ein Computer irgendwo sichtbar sein muss, ein* **KARTENLESEGERÄT**, *mit dem die Informationen von der Chipkarte gelesen werden.*
Der Kartenleser, ein USB-Gerät von Towitoko, wurde auseinandergebaut, sowie ein altes Mobiltelefon von Nokia. Die SIM-Kartenhalterung aus dem Telefon wurde, über Kabel, an den Kartenleser gelötet und dann unter das Joypad geschraubt, was nötig war, da die eigentliche Kartenhalterung des Kartenlesers nicht sehr ansehnlich war. Zum Starten des Lesevorgangs wurde noch ein Schalter benötigt, der kurzer Hand dem Joypad entliehen wurde.

A02

PROGRAMMSTRUKTUR

DIE PROGRAMMIERUNG DER ARBEIT *ist sehr aufwendig, deshalb ist es absolut notwendig, die Struktur vorher zu visualisieren, um mich während der Programmierung an den Graphiken orientieren zu können. Neben der eigentlichen Programmstruktur habe ich auch verschieden Probleme visualisiert, um ein besseres Verständis zu gewinnen.*

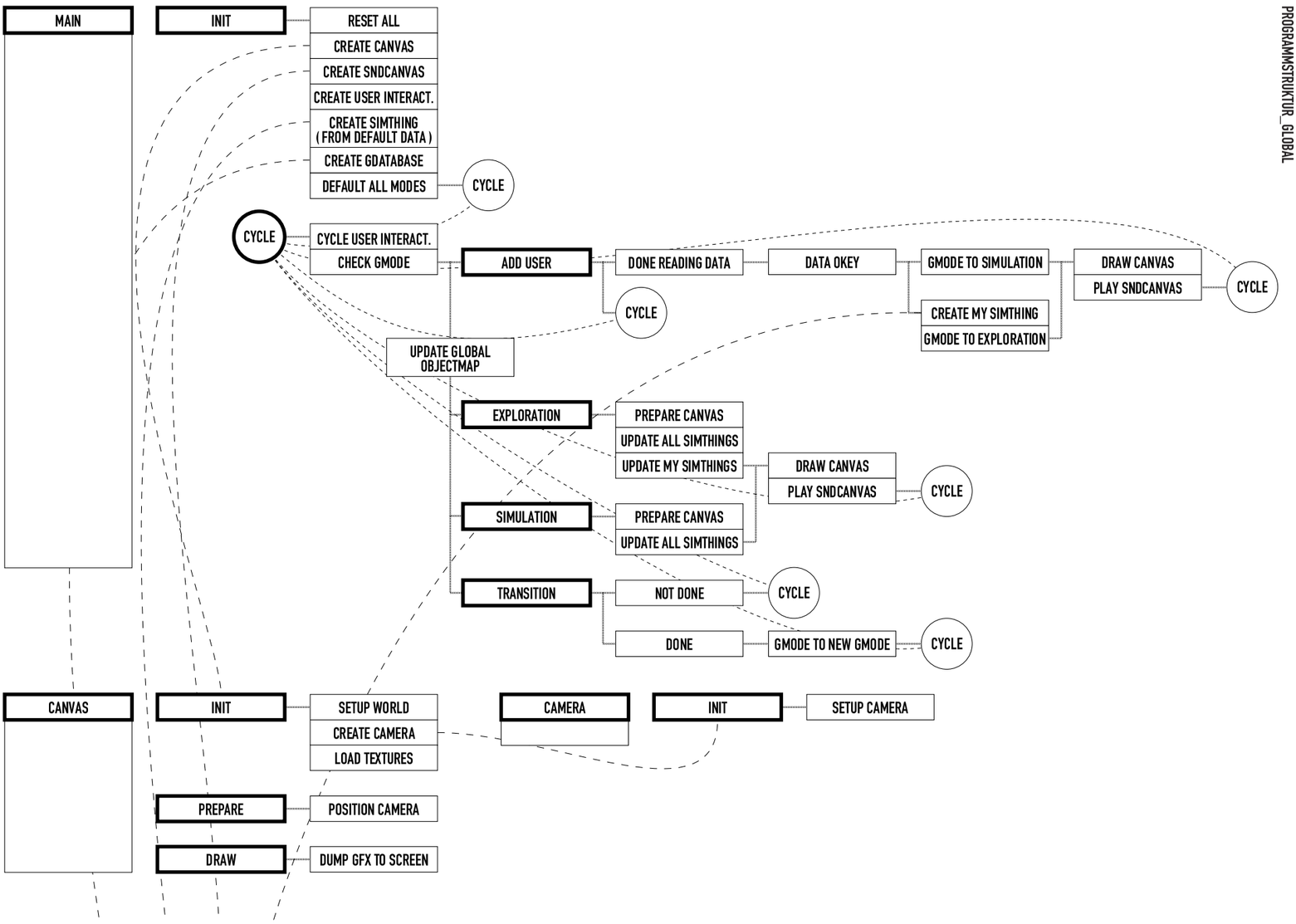
OBJEKTE_UND_MODES

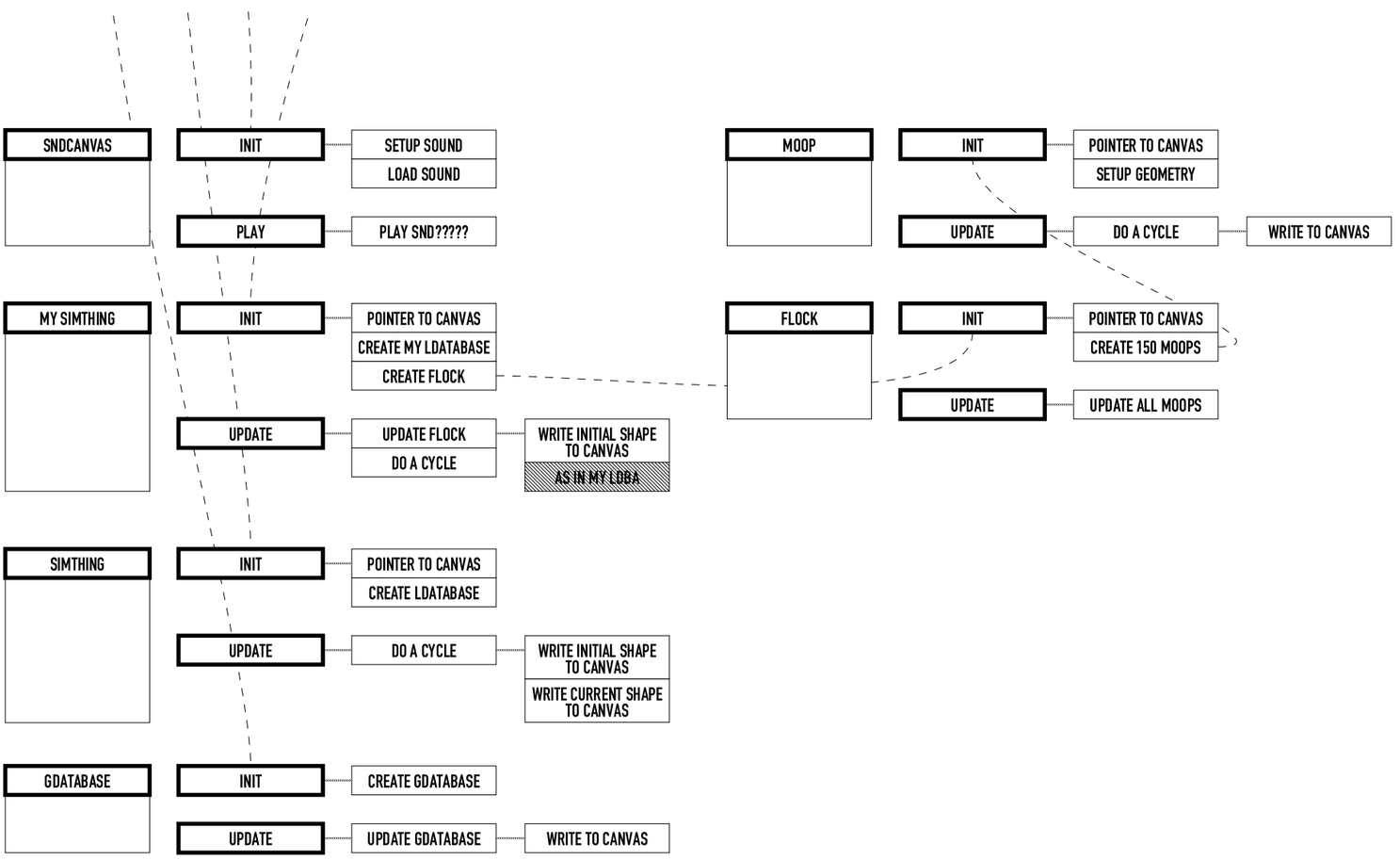


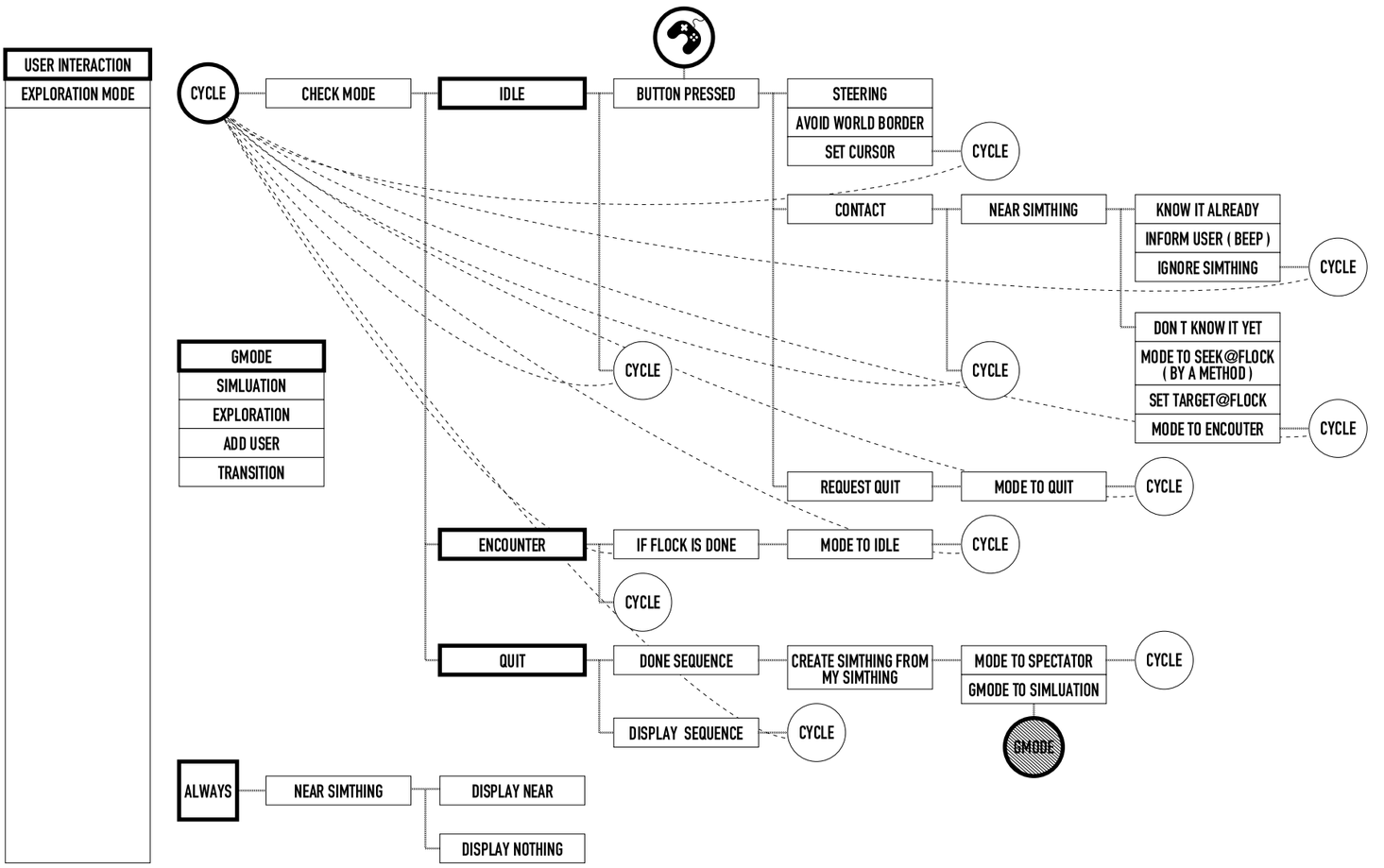
OBJEKTE *Das Programm ist so strukturiert, dass es verschieden Objekte gibt, die über bestimmte Känale miteinander kommunizieren können, um Informationen über Position, Zustand etc auszutauschen. In der Skizze auf der nächsten Seite wird eine globale Sicht dargestellt. Es gibt folgende Objekte -›* **MAIN** *ist das zentrale element, dass zB den Zeittakt vorgibt in dem andere Objekte einen nächsten Schritt durchführen. Ausserdem speichert main den globalen Zustand der Applikation. Es gibt 4 verschiedene globale Zustände (* **GMODE** *), in* **EXPLORATION** *befindet sich die Applikation in dem Modus in dem ein Benutzer seine eigenen Daten steuert und seine Verbindungen untersucht, in* **SIMULATION** *läuft das Programm selbständig ab und die einzelnen Objekte führen ein Eigenleben, in* **ADDUSER** *kann ein Benutzer seine SIM Karte auslesen lassen und* **TRANSITION** *stellt verschiedene Übergänge zwischen den 3 anderen modes zur Verfügung. Das Objekt* **CANVAS** *stellt einen OpenGL Kontext zur Verfügung an den alle Befehle gehen, die etwas mit der Darstellung auf dem Bildschirm zu tun haben, dh alle Objekte die etwas auf den Bildschirm zeichnen haben eine Verbindung zu diesem Objekt. sndcanvas ist ein ähnliches Objekt, nur das es die Ausgabe von Geräuschen realisiert. Das Objekt* **MYSIMTHING** *repräsentiert die Daten, die ein Benutzer aus seiner SIM Karte ausgelesen hat, es erzeugt ein weiteres Objekte;* **FLOCK** *ist die Darstellung aller maximal 150 Nummern in Form eines Schwarms, wobei die einzelnen*
Schwarm Objekte **MOOP** *heissen. Diese drei Objekte existieren nur im exploration Mode.* **ASIMTHING** *oder* **SIMTHING** *ist ein Objekt, dass die Benutzer repräsentiert, die einmal ihre Daten zur Verfügung gestellt haben.* **CARD READER** *stellt die Anbindung zu einem Kartenleser zur Verfügung.*

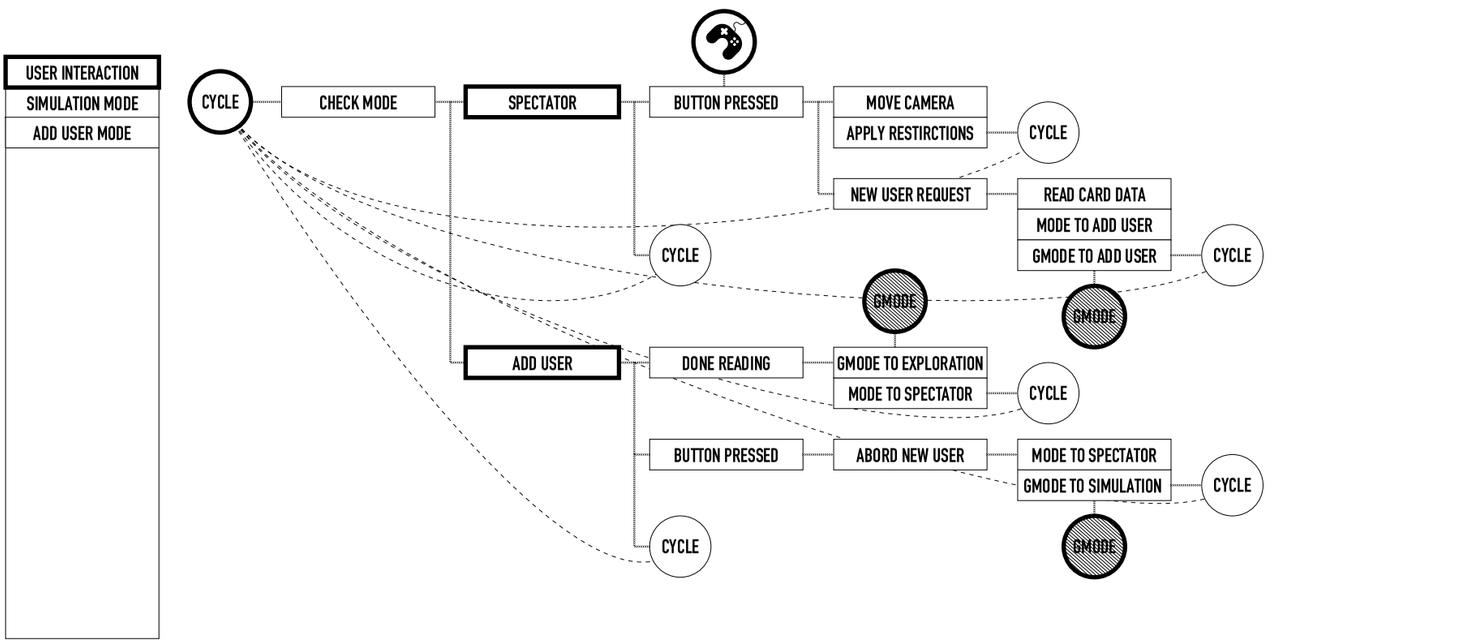
MODES *Die verschiedenen Zustände der einzelnen Objekte werden in ›modes‹ beschrieben. Das Programm ist gewissermassen eine ›state machine‹ dh ein Objekt bleibt so lange in einem ›state‹, bis dieser ausdrücklich geändert wird. In diesem Fall durch ›cycle‹ & ›mode‹ repräsentiert. Der Zustand eines Objektes kann entweder durch das Objekt selber verändert werden, wenn zB eine bestimmte Bedingung erfüllt wird, oder er wird von anderen Objekten geändert. Es gibt einen ›gmode‹, der den allgemeinen Zustand des Programms beschreibt.*

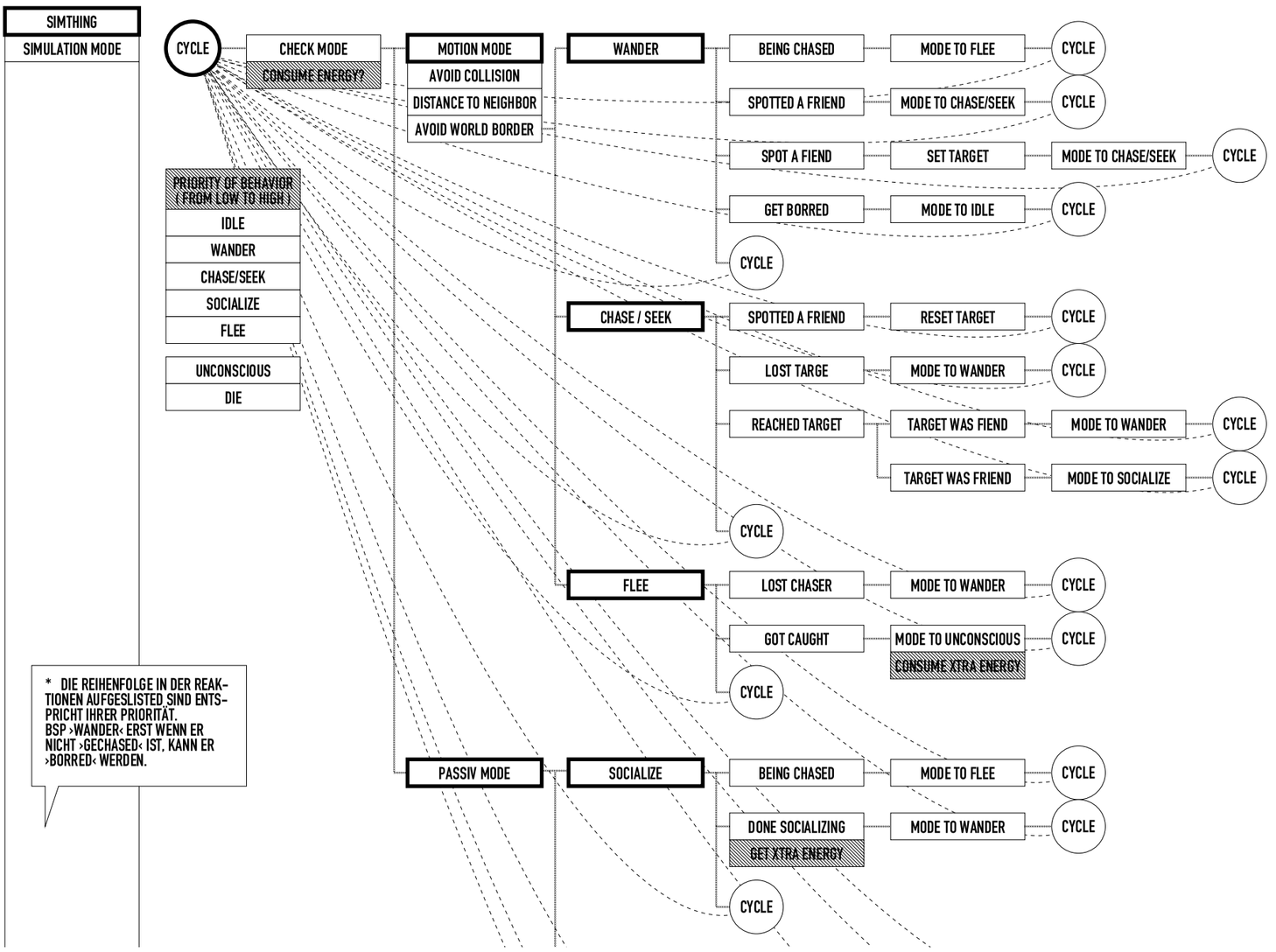
PROGRAMMSTRUKTUR_GLOBAL

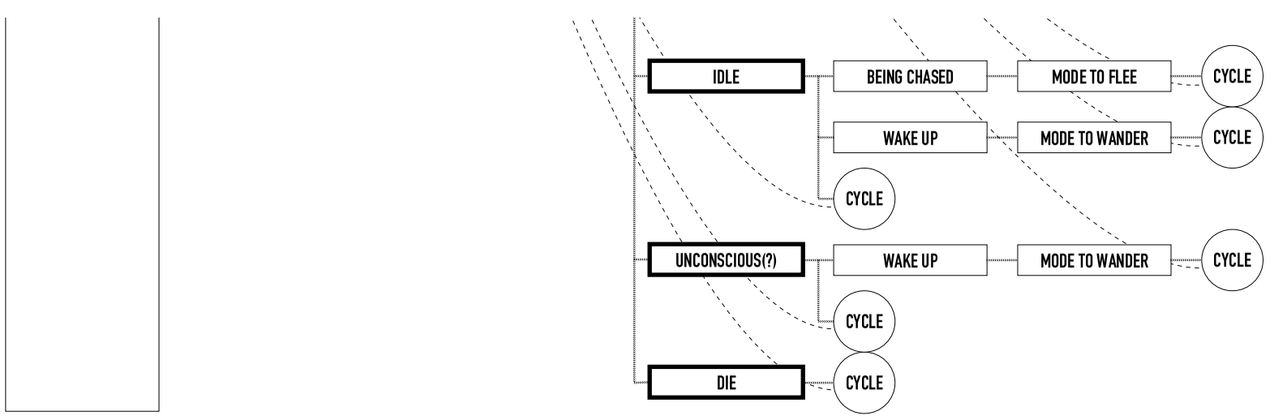


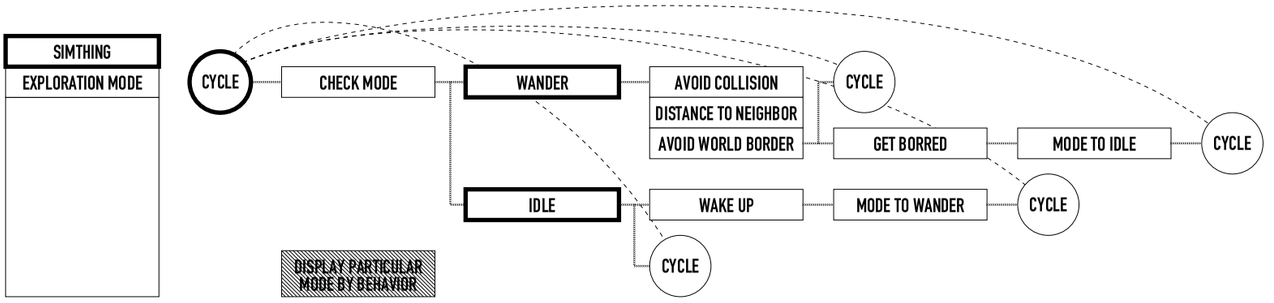


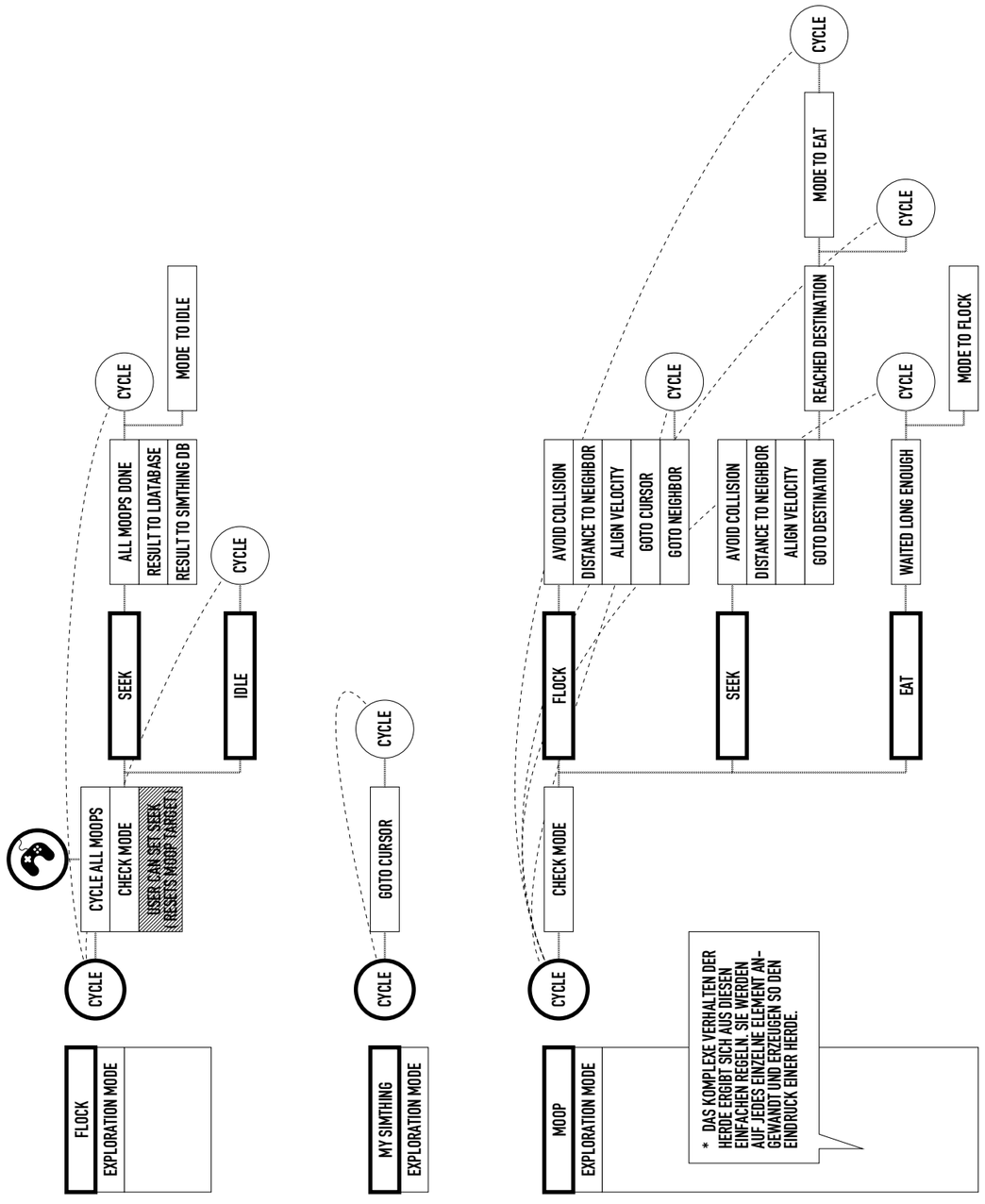

USER INTERACTION
EXPLORATION MODE
CYCLE
CHECK MODE
IDLE
BUTTON PRESSED
STEERING
AVOID WORLD BORDER
SET CURSOR
CYCLE
CONTACT
NEAR SIMTHING
KNOW IT ALREADY
INFORM USER (BEEP)
IGNORE SIMTHING
CYCLE
DON T KNOW IT YET
MODE TO SEEK@FLOCK (BY A METHOD)
SET TARGET@FLOCK
MODE TO ENCOUTER
CYCLE
CYCLE
CYCLE
GMODE
SIMLUATION
EXPLORATION
ADD USER
TRANSITION
REQUEST QUIT
MODE TO QUIT
CYCLE
ENCOUNTER
IF FLOCK IS DONE
MODE TO IDLE
CYCLE
CYCLE
QUIT
DONE SEQUENCE
CREATE SIMTHING FROM MY SIMTHING
MODE TO SPECTATOR
GMODE TO SIMLUATION
CYCLE
DISPLAY SEQUENCE
CYCLE
GMODE
ALWAYS
NEAR SIMTHING
DISPLAY NEAR
DISPLAY NOTHING


USER INTERACTION
SIMULATION MODE
ADD USER MODE
CYCLE
CHECK MODE
SPECTATOR
BUTTON PRESSED
MOVE CAMERA
APPLY RESTIRCTIONS
CYCLE
NEW USER REQUEST
READ CARD DATA
MODE TO ADD USER
GMODE TO ADD USER
CYCLE
CYCLE
GMODE
GMODE
ADD USER
DONE READING
GMODE TO EXPLORATION
MODE TO SPECTATOR
CYCLE
BUTTON PRESSED
ABORD NEW USER
MODE TO SPECTATOR
GMODE TO SIMULATION
CYCLE
CYCLE
GMODE

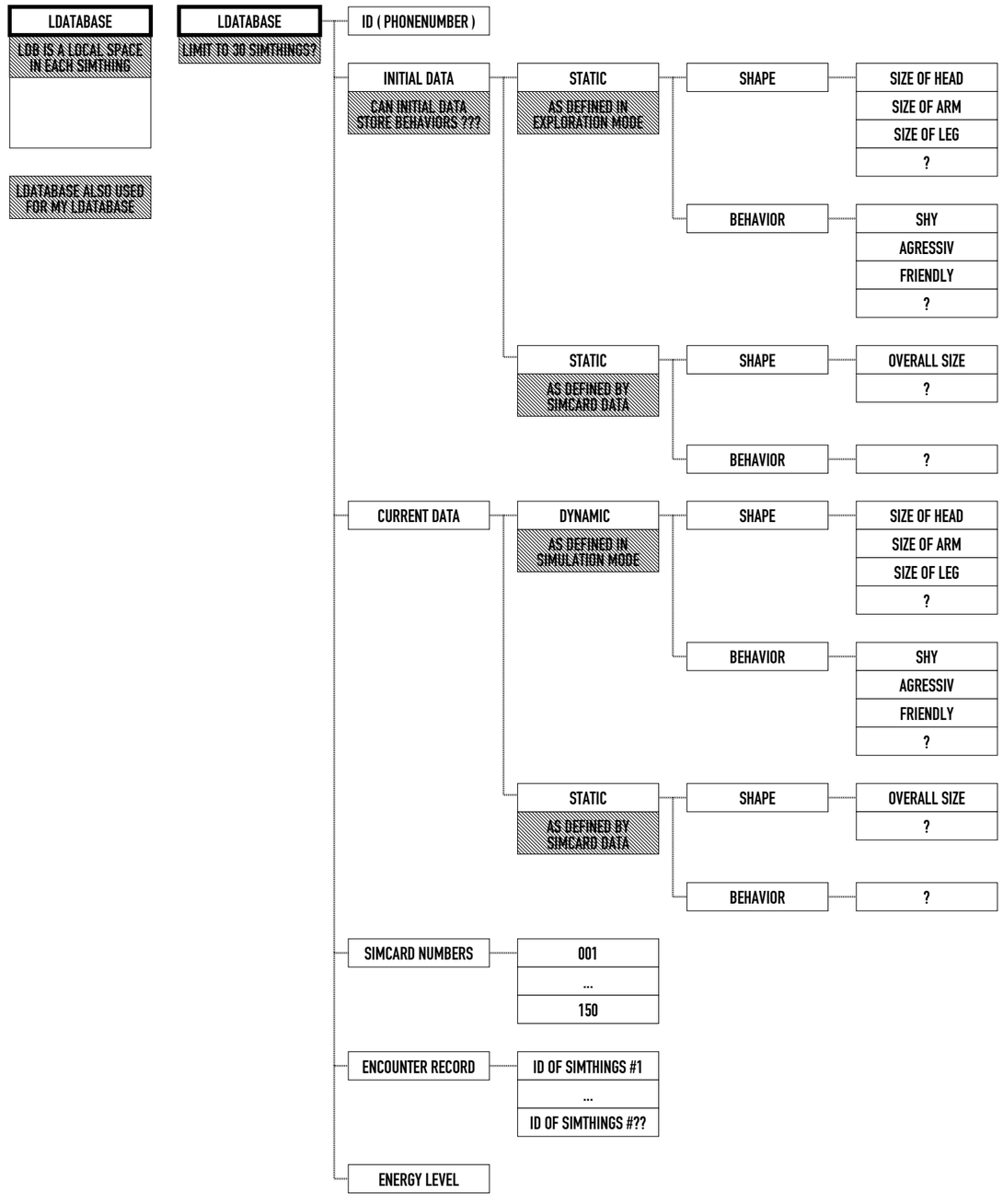

SIMTHING
SIMULATION MODE
CYCLE
CHECK MODE
CONSUME ENERGY?
MOTION MODE
AVOID COLLISION
DISTANCE TO NEIGHBOR
AVOID WORLD BORDER
PRIORITY OF BEHAVIOR (FROM LOW TO HIGH)
IDLE
WANDER
CHASE/SEEK
SOCIALIZE
FLEE
UNCONSCIOUS
DIE
WANDER
BEING CHASED
MODE TO FLEE
SPOTTED A FRIEND
MODE TO CHASE/SEEK
SPOT A FIEND
SET TARGET
GET BORRED
MODE TO IDLE
CHASE / SEEK
RESET TARGET
LOST TARGE
MODE TO WANDER
REACHED TARGET
TARGET WAS FIEND
TARGET WAS FRIEND
MODE TO SOCIALIZE
FLEE
LOST CHASER
GOT CAUGHT
MODE TO UNCONSCIOUS
CONSUME XTRA ENERGY
PASSIV MODE
SOCIALIZE
DONE SOCIALIZING
GET XTRA ENERGY
* DIE REIHENFOLGE IN DER REAK-
TIONEN AUFGESLISTED SIND ENTS-
PRICHT IHRER PRIORITÄT.
BSP ›WANDER‹ ERST WENN ER
NICHT ›GECHASED‹ IST, KANN ER
›BORRED‹ WERDEN.


IDLE
BEING CHASED
MODE TO FLEE
WAKE UP
MODE TO WANDER
UNCONSCIOUS(?)
DIE
CYCLE


SIMTHING
EXPLORATION MODE
CYCLE
CHECK MODE
WANDER
AVOID COLLISION
DISTANCE TO NEIGHBOR
AVOID WORLD BORDER
GET BORRED
MODE TO IDLE
IDLE
WAKE UP
MODE TO WANDER
DISPLAY PARTICULAR
MODE BY BEHAVIOR

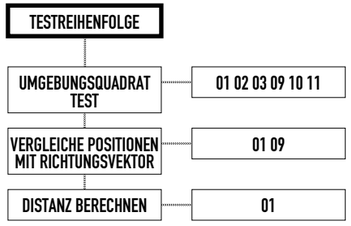


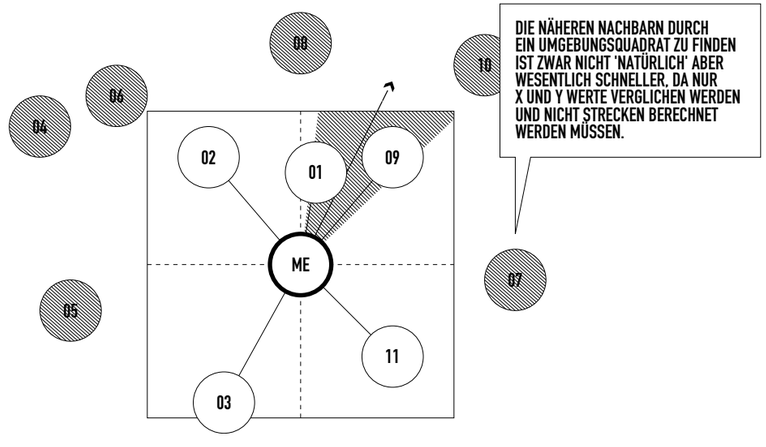
DATENSTRUKTUR EINES SIMTHING



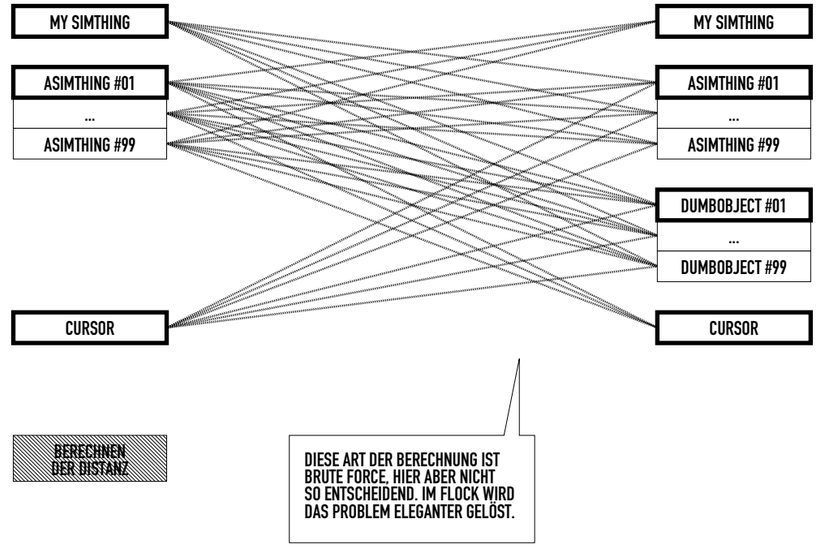
NACHBARNFINDEN

Während der Laufzeit müssen eine ganze Menge von Beziehungen zwischen den einzelnen Objekten berechnet werden. Es gibt verschiedene Möglichkeiten die Menge der Berechnungen zu optimieren. Unten auf der Seite wird gezeigt, wie möglichst effektiv die Nachbarn in der Umgebung eines Objektes gefunden werden können. Das Ergebnis ist einerseits weniger Rechenaufwand, andererseits aber auch ein realistischeres Verhalten, da in diesem Verfahren bestimmt werden kann, wie weit ein Objekt ›kucken‹ kann und was für einen Sichtwinkel es hat. So kann zB die Tatsache, das Vögel, durch ihre sehr seitlich angebrachten Augen, in einem Umfeld mit einem Winkel von 320° kucken können, allerdings wenig Tiefenwahrnehmung haben, von Bedeutung sein, wenn man das Verhalten von Vogelschwärmen benutzen möchte.
Auf der nächsten Seiten ist einerseits eine Erleuterungen zu allen nötigen Distanzberechnungen zu sehen und andererseits ein konkrete Optimierung der Berechnung von Distanzen für den ›flock‹.





DISTANCEMAP GLOBAL



DISTANCEMAP FLOCK

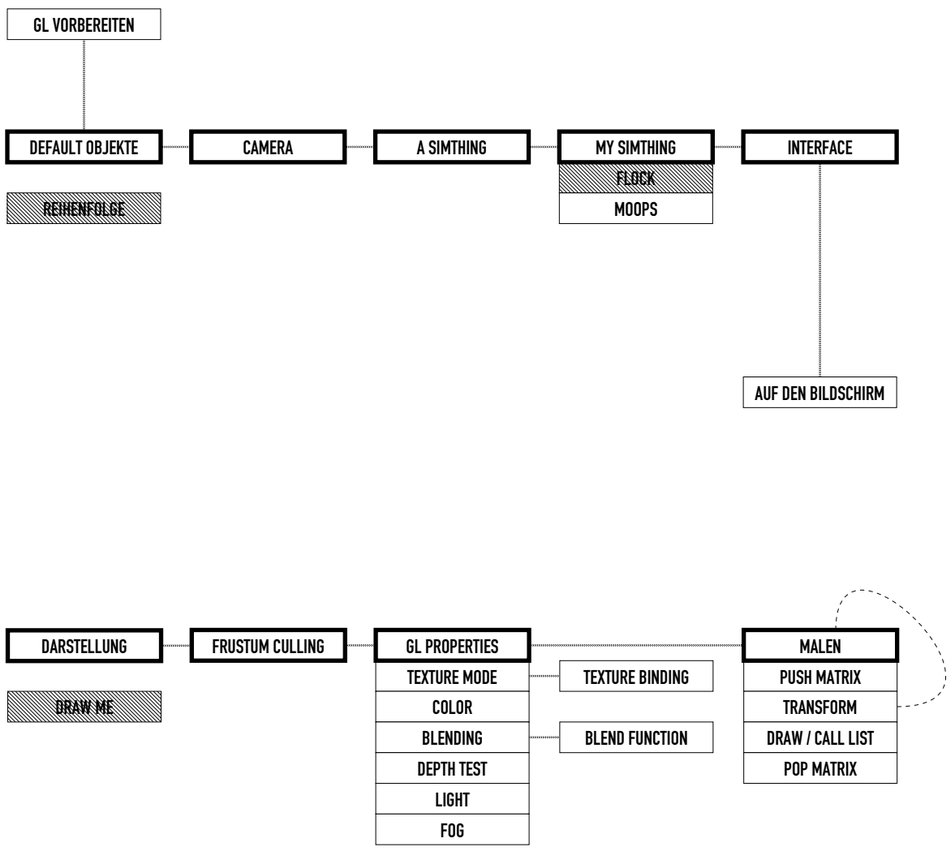
	MOOP #001	MOOP #002	MOOP #003	MOOP #004	MOOP #005	MOOP #006	MOOP #...	MOOP #150
MOOP #001		#001 – #002	#001 – #003	#001 – #004	#001 – #005	#001 – #006	#001 – #...	#001 – #150
MOOP #002			#002 – #003	#002 – #004	#002 – #005	#002 – #006	#002 – #...	#002 – #150
MOOP #003				#003 – #004	#003 – #005	#003 – #006	#003 – #...	#003 – #150
MOOP #004					#004 – #005	#004 – #006	#004 – #...	#004 – #150
MOOP #005						#005 – #006	#005 – #...	#005 – #150
MOOP #006							#006 – #...	#006 – #150
MOOP #...								#007 – #150
MOOP #150								

DIE MENGER DER BERECHNUNGEN REDUZIERT SICH SO VON
(ANZAHL*ANZAHL) AUF (!(ANZAHL-1)).
(80 * 80) = 6400
!(80 - 1) = 3160

MECHANIK DER OBJEKTDARSTELLUNG

Die Darstellung der Objekte ist strukturiert und passiert in jedem ›cycle‹ in der gleichen Reihenfolge. Bevor irgendein Objekt zeichnet, muss der ›canvas‹ vorbereitet werden. Danach positioniert die **CAMERA** *das Weltkoordinatensystem. Jedes Objekt, das eine Darstellung besitzt, berechnet seine neue Position, Erscheinung etc und schreibt diese in das ›canvas‹ objekt. Am Ende eines ›cycles‹ wird dann alles* **AUF DEN BILDSCHIRM** *gebracht.*

Die meisten Objekte besitzen eine **DRAW ME** *Methode in der ihr aktueller Zustand, meistens am Ende eines lokalen ›cycles‹, in ›canvas‹ geschrieben wird. Bei der Darstellung von Objekten müssen einige Parameter beachtet werden, wie in der untern Grafik gezeigt. In* **FRUSTUM CULLING** *wird geprüft, ob das Objekt überhaupt im sichtbaren Bereich liegt, ist dies nicht der Fall wird, das Malen des Objektes sofort beendet.* **GL PROPERTIES** *setz eine Reihe von OpenGL Eigenschaften die zur Darstellung des Objekt wichtig sind. Am Ende werden in* **MALEN** *die eigentlichen Parameter für das malen von Objekten übergeben.*



///KODE

Im Kapitel ›Kode‹ ist der komplett source-code der Arbeit abgedruckt.

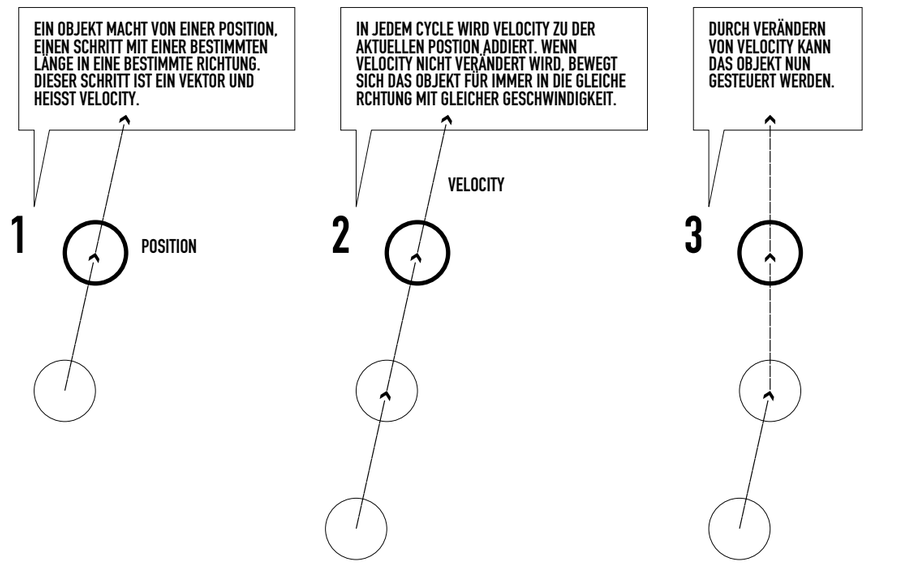
A03

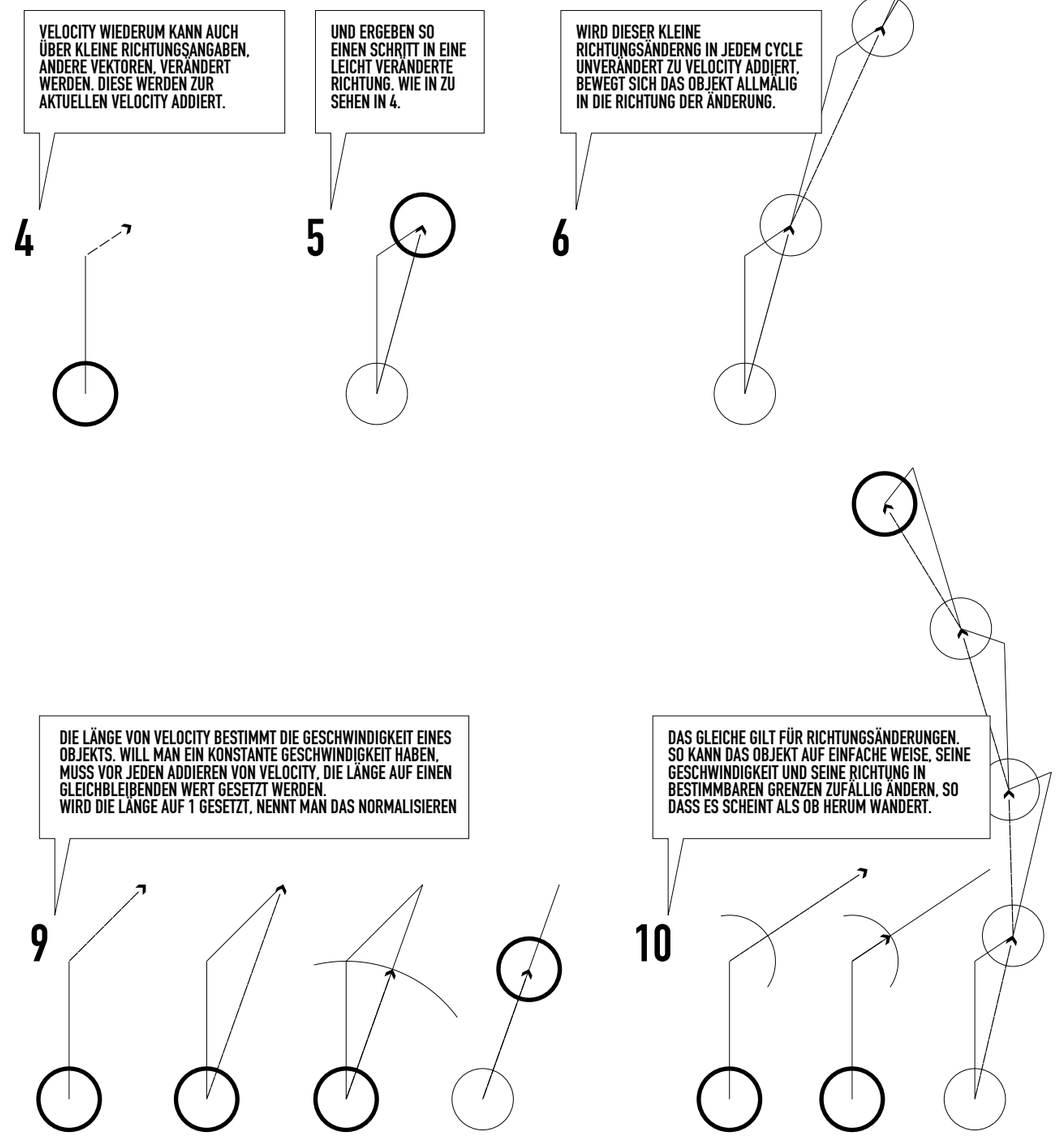
OBJEKTVERHALTEN MIT VEKTOREN

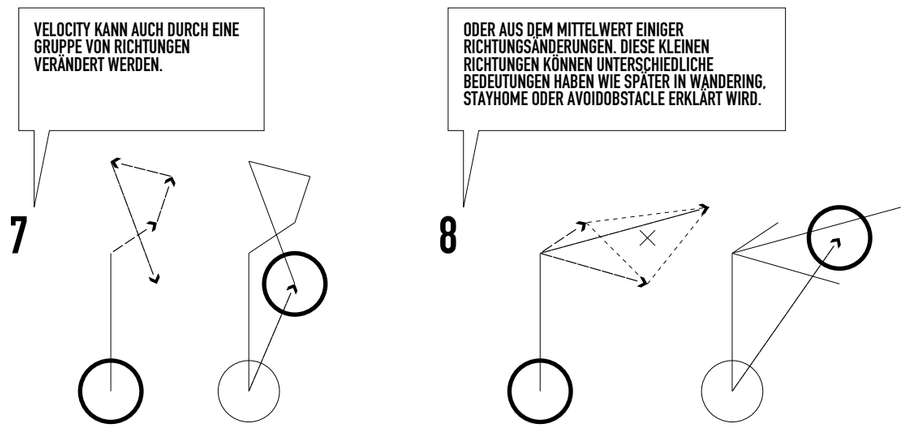
VERHALTEN MIT VEKTOREN *hört sich merkwürdig an. Das Arbeiten mit Vektoren ist allerdings unerlässlich, wenn man sich mit dem Bewegen von Objekten auseinander setzt. Deshalb sind hier die wichigsten Grundsätze und ein paar Verhalten ausführlich beschrieben und bebildert, da sie sozusagen die Grundlage der Arbeit bilden. Ein Verständnis dieser Vorgänge ist nicht entscheidend für das Verständnis der Arbeit, ganz im Gegenteil das Aufklären der Mechanik von Verhalten durch Vektormathematik entzaubert eher.*

OBJEKSTEUERUNG_ALLGEMEIN

Eine Möglichkeit Objekte zu steuern, ist das cyclische Addieren von Schritten zu einer Position. Die Schritte können auch als Richtung mit einer bestimmten Länge beschrieben werden oder als Vektoren. Dabei ist es nicht entscheidend, ob es sich um 2, 3 oder mehr-dimensionale Richtungen handelt. Die Berechnung wird meistens komplizierter, manchmal unmöglich, in den meisten Fällen ist der Weg jedoch der gleiche; Objektsteuerung durch Vektoren.





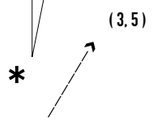


OBJEKSTEUERUNG_2DVECTORBASICS

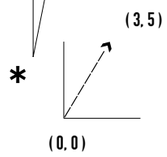
1 ADDITION ››› 2 SUBTRAKTION ››› 3 SKALAR ››› 4 LÄNGE ››› 5 NORMALISIEREN

Um mit Vektoren Objekte zu Bewegen muss man ein Paar Dinge über Vektoren wissen. Der Umgang mit diesem Wissen ist dann aber sehr einfach und intuitiv. Die mathematischen Gründe warum bestimmte Dinge mit Vektoren funktionieren sind allerdings oft wesentlich komplizierter, für den Anfang aber auch nicht so wichtig.
Das Arbeiten mit Vektoren im 2dimensionalen Raum ist wirklich ein guter Anfang, viele Dinge gelten auch für 3dimensionale Räume, manche Dinge gelten zwar für 3dimensionale Räume müssen aber aufwendiger berechnet werden. Und manche gehen gar nicht.

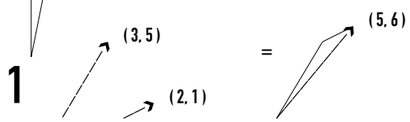
EIN VEKTOR IST EINE RICHTUNG MIT EINER BESTIMMTEN LÄNGE. IM 2D RAUM BESCHREIBT MAN EINEN VEKTOR MIT DEN KOORDINATEN X UND Y (IM 3D RAUM KOMMT NOCH Z DAZU). EIGENTLICH SCHREIBT MAN DIE BEIDEN KOORDINATEN ÜBEREINANDER, AUF DEM RECHNER ABER NICHT SO PRAKTISCH, DESHALB SIEHT EIN VEKTOR HIER SO AUS –› (3, 5)



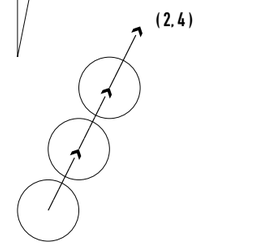
EIN VEKTOR FÄNGT IMMER BEI NULL AN –› (0, 0). AUCH URSPRUNG GENANNT. WILL MAN DEN GLEICHE VEKTOR AN EINEM ANDEREN ORT ANFANGEN LASSEN, MUSS MAN EINE ADDITION DURCHFÜHREN.



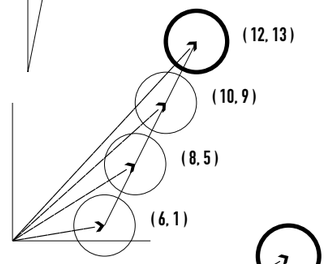
ADDITION
ZWEI VEKTOREN ZU ADDIEREN, BEDEUTET SIE ANEINANDER ZU HÄNGEN, ODER AUCH DIE KOORDINATEN ZU ADDIEREN.
(3+2, 5+1) = (5, 6) ODER ANDERS FORMULIERT
(3, 5) + (2, 1) = (5, 6)



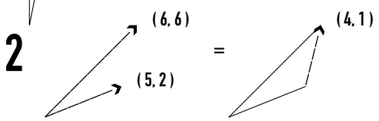
ADDIERT MAN IMMER WIEDER DEN GLEICHEN VEKTOR ZU DER POSITION EINES OBJEKTES, BEWEGT SICH DAS OBJEKT AUF EINER GERADEN LINIE.



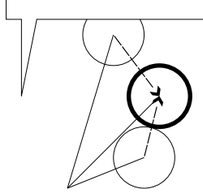
DABEI KANN DIE AKTUELLE POSITION EINES OBJEKTES AUCH ALS VEKTOR GESEHEN WERDEN.
(6, 1) + (2, 4) = (8, 5)
(8, 5) + (2, 4) = (10, 9)
(10, 9) + (2, 4) = (12, 13)



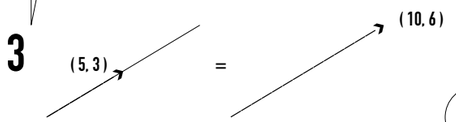
SUBTRAKTION
ZWEI VEKTOREN ZU SUBTRAHIEREN GIBT EINEN NEUEN VEKTOR, DER VON DEM ENDE DES EINEN VEKTORS ZUM ENDE DES ANDEREN VEKTORS GEHT.
(6–5, 6–2) = (1, 4) ODER ANDERS FORMULIERT
(6, 6) – (5, 2) = (1, 4)



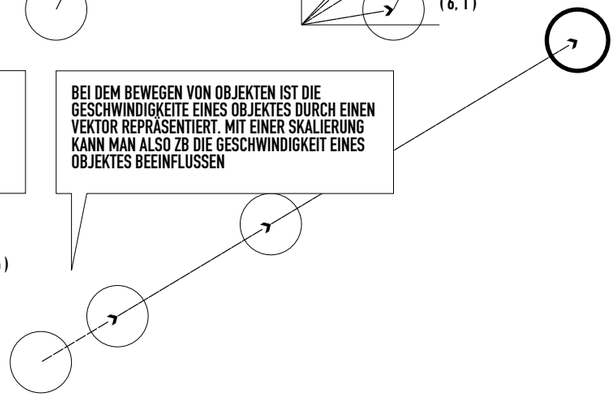
BEI DEM BEWEGEN VON OBJEKTEN KANN EINE SUBTRAKTION BENUTZT WERDEN UM HERAUSZUFINDEN, IN WELCHER RICHTUNG EIN OBJEKT IM VERHÄLTNIS ZU EINEM ANDEREN LIEGT.



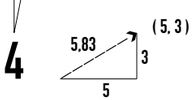
SKALAR
EIN VEKTOR IST EINE RICHTUNG MIT EINER BESTIMMTEN LÄNGE. UM DIESE LÄNGE ZU VERÄNDERN MUSS MAN DEN VEKTOR SKALIEREN.
(2*5, 2*3) = (10, 6) ODER ANDERS FORMULIERT
2*(5, 3) = (10, 6)



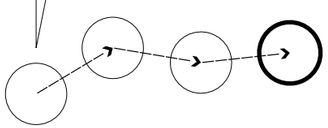
BEI DEM BEWEGEN VON OBJEKTEN IST DIE GESCHWINDIGKEITE EINES OBJEKTES DURCH EINEN VEKTOR REPRÄSENTIERT. MIT EINER SKALIERUNG KANN MAN ALSO ZB DIE GESCHWINDIGKEIT EINES OBJEKTES BEEINFLUSSEN



LÄNGE
DIE LÄNGE EINES VEKTORS LÄSST SICH MIT DEM SATZ VON PYTHAGORAS BERECHNEN –› A*A + B*B = C*C
ALSO IST DIE LÄNGE D EINES VEKTORS D = √(X*X + Y*Y)
√(5*5 + 3*3) = √34 ≈ 5,83

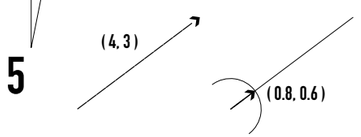


BEI DEM BEWEGEN VON OBJEKTEN KANN DIE LÄNGE DIE GESCHWINDIGKEIT EINES OBJEKTES SEIN. UNABHÄNGIG VON DER RICHTUNG, BEWEGT SICH EIN SOLCHES OBJEKT MIT KONSTANTER GESCHWINDIGKEIT. MÖCHTE MAN EIN OBJEKT MIT KONSTANTER GESCHWINDIGKEIT BEWEGEN, MUSS MAN NUR DARAUF ACHTEN, DASS DIE LÄNGE DES RICHTUNGSVEKTORS (VELOCITY) KONSTANT BLEIBT



NORMALISIEREN
EINEN VEKTOR ZU NORMALISIEREN BEDEUTET, SEINE LÄNGE AUF 1 ZU SETZEN. IM GRUNDE IST DAS NORMALISIEREN EINE KOMBINATION AUS LÄNGE BERECHNEN UND SKALIEREN. MAN BRAUCHT NICHTS ANDERES ZU TUN ALS DEN VEKTOR MIT DEM KEHRWERT SEINER LÄNGE ZU SKALIEREN.

DIE LÄNGE VON (4, 3) IST √(4*4 + 3*3) = 5
ALSO MÜSSEN DIE KOORDINATEN JEWEILS MIT DEM KEHRWERT VON 5 MULTIPLIZIERT WERDEN,
1/5 = 0.2
(4*0.2 / 3*0.2) = (0.8, 0.6)

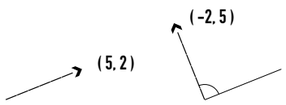


OBJEKSTEUERUNG_2DVECTORBASICS

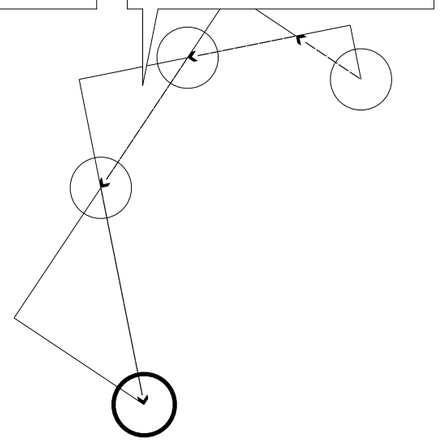
Diese Beispiele beschreiben nur die Grundlagen zum Bewegen von Objekten mit Vektoren. Es gibt noch wesentlich komplexere Möglichkeiten Objekte zu bewegen, aber als ein Anfangspunkt sind diese Beschreibungen absolut hilfreich.

6

KREUZPRODUKT
IM 3D RAUM IST DAS KREUZPRODUKT VON ZWEI VEKTOREN EIN VEKTOR DER RECHTWINKELIG ZU BEIDEN IST. IM 2D RAUM IST ES DER VEKTOR DER RECHTWINKELIG ZU EINEM ANDEREN VEKTOR IST. DAS KREUZPRODUKT IST EIN SPEZIELLE ART DER VEKTORROTATION!
UM DAS KREUZPRODUKT ZU BERECHNEN MULTIPLIZIERT MAN Y MIT -1 UND VERTAUSCH DANN X UND Y.
(5, 2) -› (-2, 5)

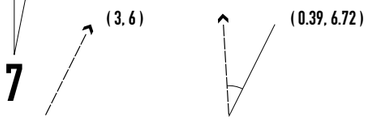


ADDIERT MAN DAS KREUZPRODUKT VOM LETZTEN SCHRITT MIT DEM LETZEN SCHRITT ZUM NÄCHSTEN SCHRITT, BEWEGT SICH DAS OBJEKT IN EINER SPIRALE. WÜRDE MAN JEDEN NEUEN SCHRITT VORHER NORMALISIEREN WÜRDE SICH DAS OBJEKT IM KREIS BEWEGEN.



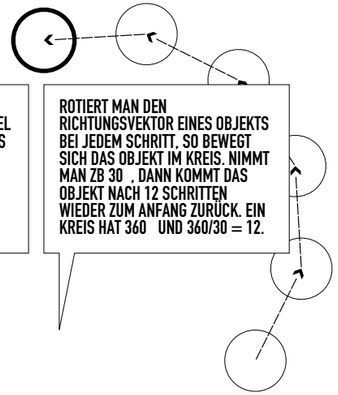
7

ROTATION
EIN VEKTOR LÄSST SICH UM EINEN BELIEBIGEN WINKEL ROTIEREN. UM EINEN VEKTOR UM DEN WINKEL A ZU ROTIEREN BRAUCHT MAN EINE KOMPLIZIERTE FORMEL -›
X = X * COS(A) - Y * SIN (A)
Y = X * SIN(A) + Y * COS (A)
UND FÜR DEN VEKTOR (3, 6) UND EINEN WINKEL VON 30 SIEHT DAS DANN SO AUS -›
X = 3 * COS (30) - 6 * SIN (30) = 3 * 0.87 - 6 * 0.50 = 0.39
Y = 3 * SIN (30) + 6 * COS (30) =3 * 0.50 + 6 * 0.87 = 6.72



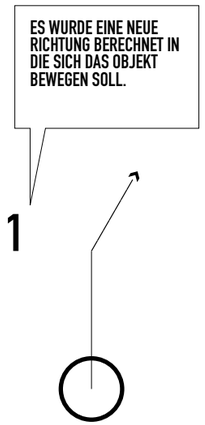
PS (NIMMT MAN 90 ALS ROTATONSWINKEL KOMMT MAN AUF DAS KREUZPRODUKT, ALLERDINGS NUR IM 2DIMENSIONALEN RAUM)

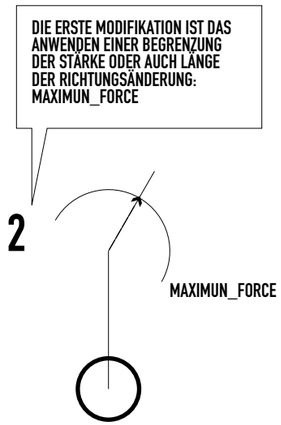
ROTIERT MAN DEN RICHTUNGSVEKTOR EINES OBJEKTS BEI JEDEM SCHRITT, SO BEWEGT SICH DAS OBJEKT IM KREIS. NIMMT MAN ZB 30 , DANN KOMMT DAS OBJEKT NACH 12 SCHRITTEN WIEDER ZUM ANFANG ZURÜCK. EIN KREIS HAT 360 UND 360/30 = 12.

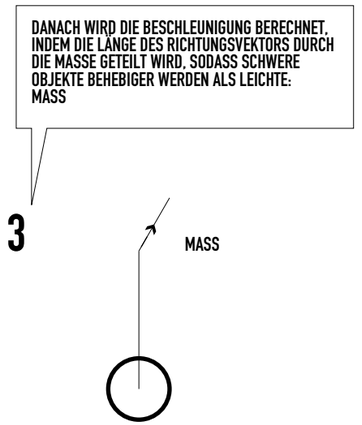


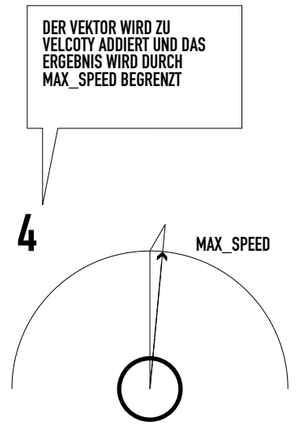
OBJEKSTEUERUNG_PHYSIK

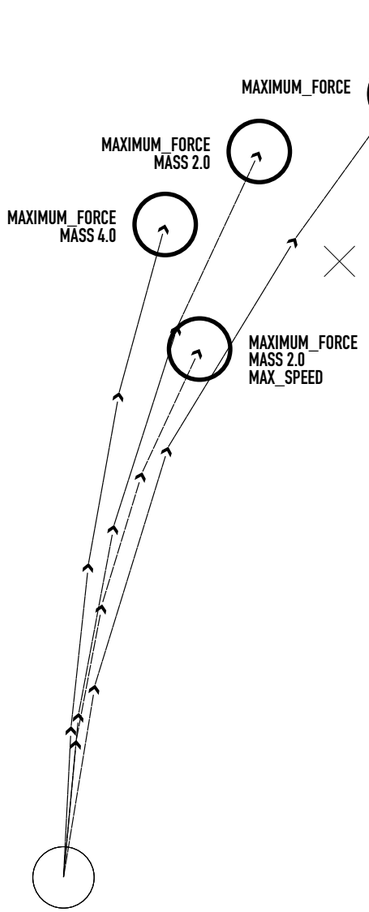
Durch das Vorschlagen von verschiedenen neuen Richtungen kann die Bewegung eines Objektes beinflusst werden. Durch das Gewichten dieser verschiedenen Vorschläge, kann ein bestimmtes Verhalten oder eine bestimmte Absicht in den Vordergrund gestellt werden. Es gibt allerdings auch noch andere Kräfte, zB physikalische, die auf das Objekt einwirken können. Durch das Anwenden von physikalischen Gesetzen kann dem Objekt ein realistischeres Verhalten verleihen.

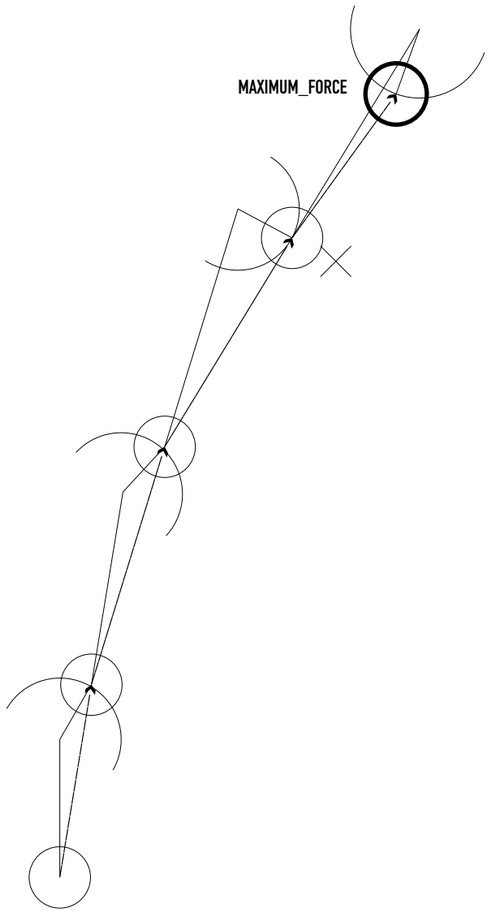


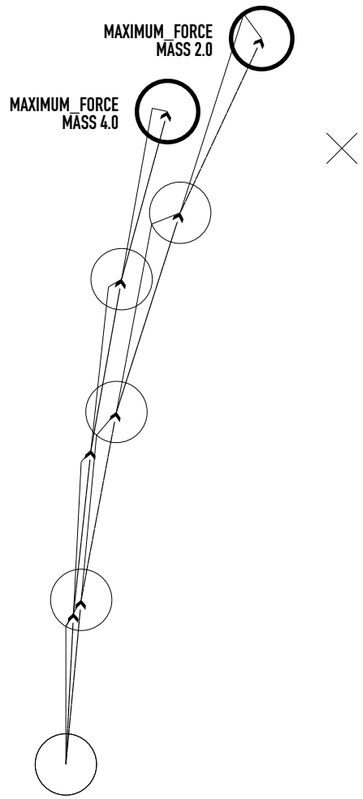


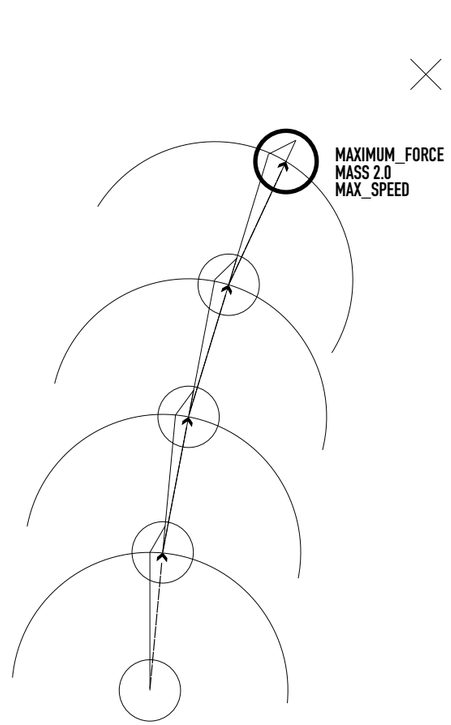






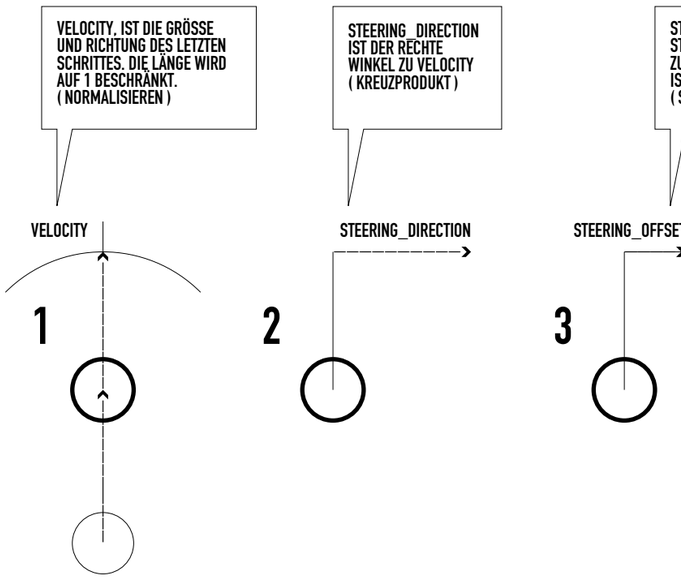


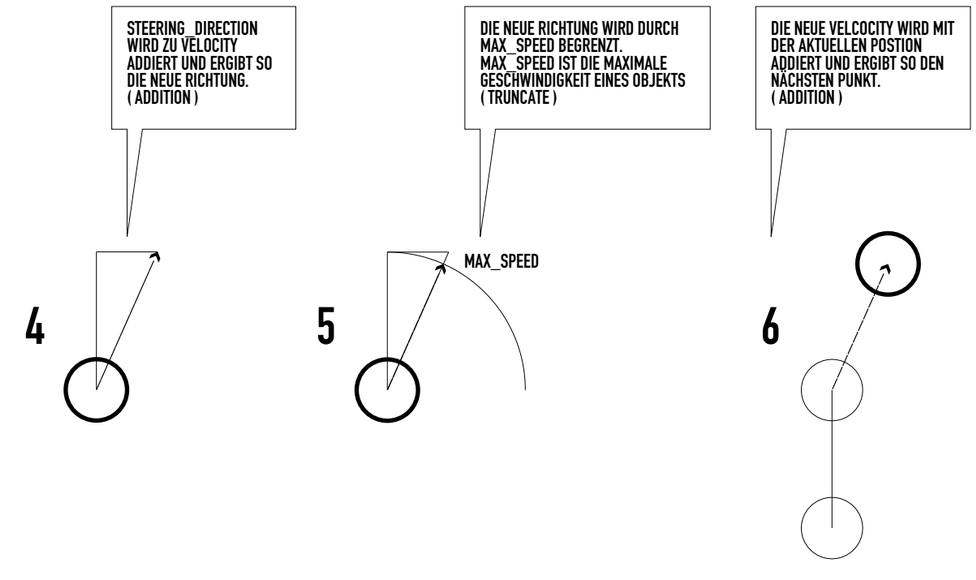




OBJEKSTEUERUNG_WANDERING

Wandering ist eine Art von zufälliger Bewegung, die zwar zufällig aussieht, aber nicht ruckelig ist sondern organisch und gleichmässig. Um das zu erreichen wird die Bewegung des Objekts in jedem Schritt leicht verändert, so dass das Objekt leicht hin und her lenkt aber nie Ruckhaft. Wandering ist ein beiläufige Art der Bewegung, ohne genaues Ziel.





KODE

```
package simStrings;

import org.lwjgl.*;
import org.lwjgl.opengl.*;
import org.lwjgl.input.*;

import java.lang.Math;
import java.util.*;

public class AlienString {
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // variables
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    public Canvas pCanvas;
    public SimStringMain pSimStringMain;

    public int ID;
    public int[] IDNumber;
    public String[] numbers;

    private String localMode = new String();
    public String pMode;

    public int localTime = 0;
    public int passingTime = 0;
    public int durationOfSwing = 100;
    public int durationOfOffswing = 250;
    public int durationOfChallenge = 500;
    public int durationOfIdle = 100;

    private String DreamNumber = new String();

    public Rotation pContact = null;

    public float height = 5.0f;
    public Vector3f position = new Vector3f((float)(Math.random()*Settings.TERRAIN_SIZE*2-Settings.TERRAIN_SIZE),
                                            (float)(Math.random()*Settings.TERRAIN_SIZE*2-Settings.TERRAIN_SIZE));

    public Vector3f velocity = new Vector3f(0,
                                            0.0f,
                                            0.0f);

    // distance maps
    public float fDS = 100.0f;
    public float testArea = 5.0f;
    public float minDistance = 1.0f;

    // relation and data
    public int relationship = 0; // relationship;
    public int socialPower; // is defined in update neighbors

    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // behavior specific
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    //      behavior
    private Vector3f new_steering = new Vector3f();
    //      behavior wander
    private Vector3f wander_velocity = new Vector3f();
    private float steering_offset = 0.0f;
    private float steering_change = 0.0005f;
    private float steering_limit = 0.005f;
    private float random_velocity_offset = 0.0f;
    private float random_velocity_change = 0.001f;
    private float random_velocity_limit = 0.005f;
    private float max_speed = 0.05f;
    private Vector3f avoidObject_velocity = new Vector3f();
    //      ufosetting
    private float turnStyle;

    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // applySteering
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    private float max_force = 0.01f;
    private float mass = 10.0f;
    private float max_speed = 0.05f;
    private float min_speed = 0.005f;

    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // other objects
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    //      duelAlien
    public AlienString nearDuelAlien = null;
    public float distanceToNearDuelAlien = 0;
    //      alienString
    public AlienString nearAlienString = null;
    public float distanceToAlienString = 0;
    //      myAlienString
    public AlienString nearMyAlienString = null;
    public float distanceToMyAlienString = 0;

    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // constructors
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    public AlienString(Canvas tCanvas, SimStringMain tSimStringMain, int tID,
                       String tIDNumber, String[] tNumbers) {
        ////////////////////////
        // objects
        ////////////////////////
        ID = tID;
        pCanvas = tCanvas;
        pSimStringMain = tSimStringMain;
        IDNumber = tIDNumber;
        numbers = tNumbers;

        ////////////////////////
        // behavior specific
        ////////////////////////
        if (Settings.RELATION_DEBUG)
            System.out.println("+ starting clothingBox.alienString #" + ID +
                               " with number " + IDNumber);

        setLocalMode("wander");

    }
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // methods
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    ////////////////////////
    // cycle()
    ////////////////////////
    public void cycle(String tGMode) {
        // get the current global mode
        gMode = tGMode;

        ////////////////////////
        //      draw do lots stuff
        if (Settings.DRAW_DEBUG)
            drawConnectLines();

        ////////////////////////
        // exploration
        ////////////////////////
        exploration();
        if (gMode == "exploration") { // alienString.exploration
            explorationAll();
        }

        ////////////////////////
        // simulation
        ////////////////////////
        if (gMode == "simulation") {
        } else if (gMode == "exploration") {
            explorationAll(); // it's supposed to have it a own method / SPAGHETTI CODE
        }

        else {
            if (Settings.RELATION_DEBUG)
                System.out.println("*** Error in alienString.gMode = " + gMode);
        }
        localTimes++;
    }

    ////////////////////////
    // exploration
    ////////////////////////
    private void exploration() {
        ////////////////////////
        //      check the local modes
        ////////////////////////
        // wander
        ////////////////////////
        if (localMode == "wander") {
            // if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
            //                                      position.z, height)) {
            //     explorationRenderDraw();
            // }
            ////////////////////////
            // peer
            ////////////////////////
            else if (localMode == "peer") { // alienString.exploration.peer
                explorationPeer();
                //      alien around depending on numberOfNeighbors ???
                //      might change localMode to exc wander
            }
            ////////////////////////
            // challenge
            ////////////////////////
            else if (localMode == "challenge") { // alienString.exploration.challenge
                explorationChallenge();
            }
            ////////////////////////
            // flee
            ////////////////////////
            else if (localMode == "flee") { // alienString.exploration.flee
                if (passingTime >= localTime) {
                    //      go back to wandering mode
                    setLocalMode("wander");
                }
                else {
                    //      perform some kinner action
                    //      maybe change the direction of challenger
                    setDirection(0);
                    new_steering.setBehavior_flee(pContact.pAlienString.position());
                    applySteeringTime_Steering();
                }

                if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                                     position.z, height)) {

                    //      get a smoothing sizin of the sphere
                    //      get the current time
                    float currentTime = durationOfSwing - (passingTime - localTime);
                    double myScale = Math.pow((double)(2.0f * currentTime - ((float)durationOfSwing)/(float)durationOfSwing), 4.0);
                    if (myScale < 0.0f) myScale = 0.0f - (float)myScale;
                    drawObject((float)myScale*10.0,
                               1.0f,
                               false, true, true);
                }
            }
            ////////////////////////
            // standstill
            ////////////////////////
            else if (localMode == "standstill") { // alienString.exploration.standstill
                if (passingTime >= localTime) {
                    //      go back to wandering mode
                    setLocalMode("wander");
                }
                else {
                    //      perform some kinner action
                    //      maybe change the direction of challenger
                    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                                         position.z, height)) {

                        explorationStandstillDraw();
                    }
                }
            ////////////////////////
            // swerfriend
            ////////////////////////
            else if (localMode == "swerfriend") { // alienString.exploration.swerfriend

                // --- looking around and performing some crazy action
                if (Settings.RELATION_DEBUG)
                    explorationSwerfriend();

            }
            ////////////////////////
            // idle
            ////////////////////////
            else if (localMode == "idle") { // alienString.exploration.idle
                if (passingTime >= localTime) {
                    //      remain idle or go wandering
                    if (Math.random() > 0.5)
                        setLocalMode("wander");
                    else {
                        if (Settings.RELATION_DEBUG)
                            System.out.println("[AlienNumber = " + pos2222! (ctl!) ");
                        passingTime = localTime + durationOfIdle;
                    }
                }
                //      perform some idle action
                if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                                     position.z, height)) {

                    explorationIdleDraw();
                }
            ////////////////////////
            // die
            ////////////////////////
            else if (localMode == "die") { // alienString.exploration.die
                //      remove alien from list and set localMode to dead for some time you DIE! and be removed from list
                if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                                     position.z, height)) {

                    explorationDieDraw();
                }
            }
            else {
                if (Settings.RELATION_DEBUG)
                    System.out.println("**** Error in alienString.exploration = " +
                                       localMode);
            }
        }

        ////////////////////////
        // setPhysicsProperties
        ////////////////////////
        private void setPhysicsProperties(String mode) {
            if (mode == "wander" || mode == "flee") {
                //      behavior wander
                wander_velocity = new Vector3f();
                steering_offset = 0.0f;
                steering_change = 0.001f;
                steering_limit = 0.005f;
                random_velocity_offset = 0.0f;
                random_velocity_change = 0.01f;
                random_velocity_limit = 0.005f;
                //      applySteering
                max_speed = 0.05f;
                mass = 5.0f;
                max_force = 0.1f;
                min_speed = 0.01f;
            }
            else if (mode == "peer") {
                //      behavior wander
                wander_velocity = new Vector3f();
                steering_offset = 0.0f;
                steering_change = 0.001f;
                steering_limit = 0.005f;
                random_velocity_offset = 0.0f;
                random_velocity_change = 0.01f;
                random_velocity_limit = 0.005f;
                //      applySteering
                max_force = 0.1f;
                mass = 10.0f;
                max_speed = 0.02f;
                min_speed = 0.01f;
            } else if (mode == "swerfriend"){
                //      behavior wander
                wander_velocity = new Vector3f();
                steering_offset = 0.0f;
                steering_change = 0.001f;
                steering_limit = 0.005f;
                random_velocity_offset = 0.0f;
                random_velocity_change = 0.01f;
                random_velocity_limit = 0.005f;
                //      applySteering
                max_force = 0.1f;
                mass = 10.0f;
                max_speed = 0.02f;
                min_speed = 0.01f;
            }
        }
        ////////////////////////
        // explorationSwerfriend
        ////////////////////////
        private void explorationSwerfriend() {
            if (nearMyAlienString != null && nearMyAlienString == null) {
                if (nearDuelAlien != null && nearMyAlienString == null) {
                    if (passingTime > localTime) {
                        //      get vector to friend
                        turnStyle = (int)Math.random() * 2 - 1;

                    }
                    else {
                        if (localTime == passingTime) {
                            //      if there is still no one there
                            //      set localMode to go to wandering mode
                            setLocalMode("wander");
                            passingTime = 0;
                        }
                        else {
                            // !!BRUTE FORCE actually needs to be done only once :)
                            //      get vector to friend
                            Vector3f vectorToFriend = new Vector3f();
                            vectorA.sub(pContact.pAlienString.position, position);
                            //      get orientation
                            float orientation = -(float) Math.atan2( -vectorA.x, vectorA.z);
                            float distanceTo = vectorA.length()*0.1f;

                            //      look at friend
                            //      calculate new orientation with turning speed
                            orientation += (position%orientation - orientation) * 0.1;
                        }
                    }
                    explorationSwerfriendDraw(true);
                }
                //      if (nearMyAlienString != null &&
                else if (nearMyAlienString != null) {
                    if (nearDuelAlien != null &&
                        (nearDuelAlien == null &&
                         distanceToDuelAlien > distanceToMyAlienString)
                        &&
                        (nearDuelAlien == null &&
                         distanceToAlienString > distanceToMyAlienString)
                        &&
                        (nearDuelAlien == null &&
                         distanceToMyAlienString < distanceToAlienString)) {
                        // what kind of alienString is close
                        if (nearMyAlienString != null) {
                            // it is a friend
                            if (pContact.breedsAndOwnerOf(nearMyAlienString) == null) {
                                // it is a friend
                                if (pContact.numberOfFriends < 0 || pContact.breedsYou == true) {
                                    //      pass by name to the other
                                    setLocalMode("passingBy");
                                }
                                else {
                                    //      is it a friend?
                                    setLocalMode("wander");
                                }
                            }
                            else {
                                if (pContact.numberOfFriends < 0 || pContact.breedsYou == true) {
                                    //      pass by name to the other
                                    setLocalMode("passingBy");
                                }
                                else {
                                    //      is it a friend?
                                    setLocalMode("wander");
                                }
                            }
                        } // if what kind of alienString
                        else if (nearDuelAlien != null) {
                            // it is a friend
                            if (pContact.breedsAndOwnerOf(nearDuelAlien) == null) {
                                // it is a friend
                                if (pContact.numberOfFriends < 0 || pContact.breedsYou == true) {
                                    //      pass by name to the other
                                    setLocalMode("passingBy");
                                }
                                else {
                                    //      is it a friend?
                                    setLocalMode("wander");
                                }
                            }
                        }
                    }
                    else {
                        if (Settings.RELATION_DEBUG)
                            System.out.println("error alienString.exploration.wander");
                    }
                }
            }
            ////////////////////////
            // explorationWander
            ////////////////////////
            private void explorationWander() {

                //      calculates the new object orientation
                //      and velocity
                //      if alienString.exploration.wander
                //      if there's enough, not wander
                if (nearMyAlienString != null && nearMyAlienString == null) {
                    if (nearMyAlienString.setSocialPower() == null && nearMyAlienString == null) {
                        //      perform wandering action and move it to a vector
                        new_steering.setVelocity(wander_velocity);
                        //      apply new velocity - including speed restrictns & world border restrictns
                        applySteering(new_steering);
                    }
                    //      what is the closest object
                    //      duelAlien
                    if (nearDuelAlien != null &&
                        (nearDuelAlien == null &&
                         (distanceToDuelAlien < distanceToAlienString &&
                          distanceToDuelAlien < distanceToMyAlienString)
                         &&
                         (nearDuelAlien == null &&
                          distanceToDuelAlien < distanceToMyAlienString)
                         &&
                         (distanceToDuelAlien < distanceToAlienString &&
                          distanceToDuelAlien < distanceToMyAlienString) == null)) {
                        if (localTime % 50 == 0) {
                            avoidObject_velocity = behavior_avoidObject(nearDuelAlien.position);
                        }
                        new_steering.setCalculatedObject_velocity();
                        //      apply new velocity - including speed restrictns & world border restrictns
                        applySteering(new_steering);
                    }
                    else {
                        //      look at challenger
                        //      calculate new orientation with turning speed
                        orientation += (destinationOrientation - orientation) * 0.1;
                    }
                    //      draw animated texture
                    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2, position.y, height)) {
                        drawBillboard(position.x, position.y, position.z, 1.0f, 1.0f, 1.0f,
                                      false, false, true);
                        //      draw scanner
                        Canvas.gl.enable(GL.TEXTURE_2D);
                        Canvas.gl.enable(GL.BLEND);
                        Canvas.gl.blendFunc(GL.SRC_ALPHA, GL.ONE);
                        Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[(localTime % 4) + 11]);
                        Canvas.gl.disable(GL.DEPTH_TEST);
                        Canvas.gl.color(GL.GL_1.0f, 1.0f, 1.0f);
                        Canvas.gl.disable(GL.CULL_FACE);
                        if (Settings.FOG)
                            Canvas.gl.disable(GL.FOG);
                        else
                            Canvas.gl.disable(GL.FOG);
                        //      position and rotate boundingsystem
                        Canvas.gl.pushMatrix(); // boundingsystem
                        Canvas.gl.translatef(position.x,
                                             position.y,
                                             position.z);
                        Canvas.gl.rotatef( (float) Math.toDegrees(orientation), 0.0f,
                                           1.0f, 0.0f);
                        Canvas.gl.rotatef( 90.0f, 1.0f, 0.0f);
                        Canvas.gl.pushMatrix();
                        Canvas.gl.scalef(1.5f,1.5f,1.5f);
                        Canvas.gl.drawSphere(1.0f,10,10);
                        Canvas.gl.popMatrix();

                        // draws of friends
                        if ((passingTime - localTime) * (360 / durationOfIdle) % 90) == 0) {
                            DreamNumber = numberOfInt((Math.random()*numbers.length));
                            //      System.out.println("drawing dream number " + DreamNumber + " passing of: "+passingTime);
                        }
                        // scale from small to big --> it is only not to be applied :)
                        float DreamNumberSize = 0.1f + 0.2f * ((360- (passingTime - localTime) * (360 / durationOfIdle)) % 90)*0.01f);
                        Canvas.gl.translatef( 0.0f,
                                              1.5f,
                                              (float)(Math.cos(Math.toRadians((passingTime - localTime) * (360 / durationOfIdle)) % 90)) * 1.5f + 0.5f,
                                              0.0f);
                        Canvas.gl.scalef(DreamNumberSize, DreamNumberSize, DreamNumberSize);
                        Canvas.gl.DDrint(DreamNumber, 1);
                        //      Canvas.gl.DDrint(DreamNumber, DreamNumber);
                        Canvas.gl.popMatrix(); // scale

                    }
                    ////////////////////////
                    // explorationIdleDraw
                    ////////////////////////
                    private void explorationIdleDraw() {
                        if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2, position.y,
                                                             position.z, height)) {
                            //drawFoodObject();
                        }
                    }
                    ////////////////////////
                    // drawObject
                    ////////////////////////
                    public void drawObject(float height, float scale, double spHereRatio,
                                           boolean displayNames,
                                           boolean velocityVisitor,
                                           boolean numbersOnly ) {
                        ////////////////////////
                        //      draw object
                        //      prepare drawing
                        float[] numbersArray = new float[numbers.length / 250.0f + 0.5f]; // size end for numberOfNumbers
                        Canvas.gl.disable(GL.TEXTURE_2D);
                        Canvas.gl.enable(GL.BLEND);
                        Canvas.gl.blendFunc(GL.SRC_ALPHA, GL.ONE_MINUS_SRC_COLOR);
                        Canvas.gl.color(GL.GL_1.0f, 1.0f, 1.0f);
                        Canvas.gl.disable(GL.DEPTH_TEST);
                        Canvas.gl.disable(GL.CULL_FACE);
                        if (Settings.FOG)
                            Canvas.gl.enable(GL.FOG);
                        else
                            Canvas.gl.disable(GL.FOG);
                        //      position and rotate boundingsystem
                        Canvas.gl.pushMatrix(); // main
                        Canvas.gl.translatef(position.x,
                                             position.y,
                                             position.z);
                        Canvas.gl.rotatef( -(float) Math.toDegrees(orientation),
                                           0.0f, 1.0f, 0.0f);
                        //      texture on
                        Canvas.gl.pushMatrix(); // scharten
```

PROGRAMM UND KLASSEN

Das Programm ist in verschiedene Klassen unterteilt. Eine Klasse ist ein Stück Programm, das sich mit einer bestimmten Funktion befasst. Durch dieses Konzept können Probleme separat bearbeiten werden und die Übersichtlichkeit des gesamten Programms kann erhöht werden. Aus diesem Grund gibt es hier Klassen mit ganz verschiedenen Funktionen wie zB Daten von der Festplatte lesen, Grafik darstellen oder Eingabegeräte abfragen.
Ausserdem ist der Kode natürlich in Spaghettiform und könnte an unzähligen Stellen optimiert werden.
:)

KLASSEN

Simthing
Settings
Splash
*BitmapComponent**
SimthingMain
ASimthing
MySimthing
Flock
Moop
UserInteraction
DataReader
DataWriter
SIMCardReader
Canvas
*CCamera**
*CFrustum**
*TextureLoader**
Vector2f
*Tuple2f**

SIMTHING *Hier wird das Programm gestartet und die Splashscreen gezeigt.*

```
//
//
// simthing
// *** started programming @ 2003 03 24 CET 02:09:12
// *** d3 @ udk.berlin.de
//
//

package simthing;

import org.lwjgl.opengl.*;
import org.lwjgl.*;

public class Simthing {

    /* ------------------------------------------------------------------ */
    // methods
    /* ------------------------------------------------------------------ */

    public static void main(String[] args) {
        System.out.println("SIMTHING /// v1.1 /// MAI 2003 /// d3 paul");
        if (Settings.DEBUG) System.out.println("* simthing is born");

        // splash
        Splash splash = new Splash(1000);
        splash = null;

        // simthingMain
        SimthingMain simthingMain = new SimthingMain();

        // gl canvas
        simthingMain.canvas.init();

        // loading please Wait
        simthingMain.canvas.prepare();
        String texture[] = {
            "simthingsplash.png", "credits.png"};
        int[] pleaseWait = simthingMain.canvas.loadTextures(texture,
                                                            "mippedMap");
        Canvas.gl.enable(GL.TEXTURE_2D);
        Canvas.gl.enable(GL.BLEND);
        Canvas.gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
        Canvas.gl.color3f(1.0f, 1.0f, 1.0f);
        Canvas.gl.disable(GL.DEPTH_TEST);
        Canvas.gl.disable(GL.LIGHTING);
        if (Settings.FOG)
            Canvas.gl.enable(GL.FOG);
        else
            Canvas.gl.disable(GL.FOG);
        // credits
        Canvas.gl.pushMatrix();
        Canvas.gl.translatef(5.0f, 0.0f, 5.0f);
        Canvas.gl.bindTexture(GL.TEXTURE_2D, pleaseWait[1]);
        Canvas.gl.rotatef( -10.0f, 0.0f, 1.0f, 0.0f);
        Canvas.gl.rotatef( -90.0f, 1.0f, 0.0f, 0.0f);
        Canvas.gl.scalef(5.0f, 5.0f, 5.0f);
        //Canvas.gl.callList(Canvas.plane);
        Canvas.gl.popMatrix();
        // logo
        Canvas.gl.translatef(0.0f, 0.0f, -5.0f);
        Canvas.gl.bindTexture(GL.TEXTURE_2D, pleaseWait[0]);
        Canvas.gl.rotatef(10.0f, 0.0f, 1.0f, 0.0f);
        Canvas.gl.rotatef( -90.0f, 1.0f, 0.0f, 0.0f);
        Canvas.gl.scalef(10.0f, 10.0f, 10.0f);
        Canvas.gl.callList(Canvas.plane);
        simthingMain.canvas.gl.swapBuffers();

        // load textures
        simthingMain.canvas.prepareTextures();

        // start
        simthingMain.run();
    }
}
```

SETTINGS *Als eine Sammlung von statischen Variablen, dient diese Klasse der globalen Verwaltung von Preferenzes.*

```
//
//
// simthing.Settings
//
//

package simthing;

public class Settings {

    public static final boolean DEBUG = false;
    public static final boolean DRAW_DEBUG = false;
    public static final boolean RELATION_DEBUG = false;
    public static final boolean NOREADER = false;
    public static final boolean FOG = false;

    public static final int[] SCREEN = { 1024, 768, 32, 60};
    public static final boolean GUI_CENTERED = true;
    public static final int GUI_CONTROLLEROFFSET = 16;

    public static final String DATADIRECTORY = "data/numbers/";
    public static final float TERRAIN_SIZE = 10.0f;
    public static final float MAX_FRAMERATE = 50.0f;
    public static final String TEXTURE_PATH = "data/textures/";

    public static final int FOOSIMTHING = 100;

}
```

SPLASH *Hier wird der erste Splashscreen dargestellt.*

```
//
//
// simthing.splash
//
//

package simthing;

import java.awt.*;
import java.awt.event.*;

public class Splash extends java.awt.Window
{

    /* ----------------------------------------------------------------- */
    // variables
    /* ----------------------------------------------------------------- */

    private Image img;

    /* ----------------------------------------------------------------- */
    // constructors
    /* ----------------------------------------------------------------- */

    public Splash(int millis)
    {
        super(new Frame());
        BitmapComponent test = new BitmapComponent(Settings.TEXTURE_PATH+"splash.png");
        setSize(test.getWidth(),test.getHeight());
        this.add(test);
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation(0,0);
        pack();

        // show
        setVisible(true);
        try {
            Thread.sleep(millis);
        } catch (InterruptedException e) {
            // wait
        }
        setVisible(false);
    }
}
```

BITMAPCOMPONENT

```
//
//
// simthing.splash.BitmapComponent
// *** from *** Handbuch der Java-Programmierung / 3. Auflage / © 1998-2002 Guido Krüger
//
//

package simthing;

import java.awt.*;

public class BitmapComponent
extends java.awt.Canvas {

    /* ----------------------------------------------------------------- */
    // variables
    /* ----------------------------------------------------------------- */

    private Image img;

    /* ----------------------------------------------------------------- */
    // constructors
    /* ----------------------------------------------------------------- */

    public BitmapComponent(String fname) {
        img = getToolkit().getImage(fname);
        MediaTracker mt = new MediaTracker(this);
        mt.addImage(img, 0);
        try {
            mt.waitForAll();
        }
        catch (InterruptedException e) {
            // wait
        }
    }

    /* ----------------------------------------------------------------- */
    // methods
    /* ----------------------------------------------------------------- */

    public void paint(Graphics g) {
        g.drawImage(img, 0, 0, this);
    }

    public Dimension getPreferredSize() {
        return new Dimension(
                            img.getWidth(this),
                            img.getHeight(this)
                            );
    }

    public Dimension getMinimumSize() {
        return new Dimension(
                            img.getWidth(this),
                            img.getHeight(this)
                            );
    }
}
```

SIMTHINGMAIN *Diese Klasse ist die Hauptklasse, hier wird jeder neue ›cycle‹ gestartet und alle anderen Objekte ›getickt‹. Ausserdem wird die Grafikkarte vorbereitet und die Benutzereingabe abgefragt.*

```
//
//
// simthing.simthingMain
//
//

package simthing;

import org.lwjgl.opengl.*;
import org.lwjgl.input.*;
import org.lwjgl.*;
import java.lang.Math;

import java.util.LinkedList;
import java.util.ArrayList;

import java.nio.IntBuffer;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;

public class SimthingMain {

    /* ------------------------------------------------------------------ */
    // variables
    /* ------------------------------------------------------------------ */

    // class variables
    public static       String        gMode;
    public static       int           localTime             = 0;
    public static       boolean       finished              = false;
    // aSimthing
    public              LinkedList    aSimthingArray        = new LinkedList();
    protected           int           maxNumberOfASimthing  = 1;
    // dumbObject
    protected           int           maxNumberOfDumbObject = 500;
    public              LinkedList    dumbObjectArray       = new LinkedList();
    // mySimthing
    protected           MySimthing    pMySimthing;
    public    static    char[]        myIDNumber            = {‘*’, ‘*’, ‘*’, ‘*’, ‘*’, ‘*’, ‘*’, ‘*’, ‘*’, ‘*’, ‘*’, ‘*’};
    public    static    char[]        myPIN                 = {‘*’, ‘*’, ‘*’, ‘*’};
    // objects
    static              Canvas        canvas                = new Canvas();
    protected           transitionSetting pTransition       = new transitionSetting(0, false, “”, “”);

    // exploration follow camera
    private Vector2f followCameraPosition  = new Vector2f();
    private int      followASimthingIndex = 0;
    public static float    followCameraMinimumDistance = 10.0f;

    // transition
    private             int           enterPointer          = 0;

    // UI
    public              String        UIMode;
    static              boolean       ROTATE_LEFT;
    static              boolean       ROTATE_RIGHT;
    static              boolean       FORWARD;
    static              boolean       BACKWARDS;
    static              boolean       LEFT;
    static              boolean       RIGHT;
    static              boolean       BUTTON_01; // O ( positiv aura ) -> buttons should be marked in some way. playstation
style.
    static              boolean       BUTTON_02; // X ( negative aura )

    // var
    private             long          lastFramesTime         = Sys.getTime();
    private             int           idleTime               = 0;

    /* ------------------------------------------------------------------ */
    // constructors
    /* ------------------------------------------------------------------ */

    public SimthingMain() {

        if(Settings.DEBUG) System.out.println(“* simthing.simthingMain”);
        String[] storedASimthing = DataReader.getStored();
        for ( int i=0; i<storedASimthing.length; i++ ){
            if(Settings.DEBUG) System.out.println(“* loading “+storedASimthing[i]);
            aSimthingArray.add(new ASimthing( canvas,
                                              this,
                                              i,
                                              storedASimthing[i],
                                              DataReader.load(storedASimthing[i])));
        }
        //     add foo simthings for the looks :)
        if (Settings.FOOSIMTHING != 0){
            if (Settings.DEBUG)
                System.out.println(“* fooing “ + Settings.FOOSIMTHING + “Simthings”);
            for (int j = 0; j < Settings.FOOSIMTHING; j++) {
                // create foo ID
                String fooID = “****” + (int) (Math.random() * 10000000);
                // create foo data
                String[] fooData = new String[(int)(Math.random()*140+1)];
                for (int i = 0; i < fooData.length; i++) {
                    fooData[i] = “017” + (int) (Math.random() * 100000000);
                }
                aSimthingArray.add(new ASimthing( canvas,
                                                  this,
                                                  j+storedASimthing.length,
                                                  fooID,
                                                  fooData));
            }
        }
        //     update relation
        updateSimthingRelation();
        // default gModes
        setGMode(“simulation”);
    }

    /* ------------------------------------------------------------------ */
    // methods
    /* ------------------------------------------------------------------ */

    /* ------------------------------- */
    // updateSimthingRelation
    /* ------------------------------- */
    public void updateSimthingRelation(){
        ASimthing tASimthing;
        for (int i = 0; i < aSimthingArray.size(); i++) {
            tASimthing = (ASimthing) aSimthingArray.get(i);
            tASimthing.updateRelation();
        }
    }


    /* ------------------------------- */
    // run
    /* ------------------------------- */
    public void run() {
        setup();
        while (!finished) {
            UserInteraction.processInput();
            canvas.prepare();
            cycle();
            lastFramesTime += Sys.getTimerResolution()/Settings.MAX_FRAMERATE;
            while(lastFramesTime>Sys.getTime()){
                // wait
            }
            canvas.update();
            //
            lastFramesTime = Sys.getTime();
        }
        cleanup();
        System.exit(0);
    }

    /* ------------------------------- */
    // setup
    /* ------------------------------- */
    static void setup() {
        UserInteraction.setup();
    }

    /* ------------------------------- */
    // cleanup
    /* ------------------------------- */
    static void cleanup() {
        Keyboard.destroy();
        Canvas.killFont();
        Canvas.gl.destroy();
        Display.destroy();
    }
```

```
/* ---------------- */
// makeScreenshot
/* ---------------- */
public static void makeScreenshot() {
    // set filename
    String filename = "screenshots/"+System.currentTimeMillis()+"_"+localTime + "_" + gMode + ".png";
    // allocate memory and pointer
    IntBuffer buff = ByteBuffer.allocateDirect(Display.getWidth() *
                                               Display.getHeight() * 4).order(ByteOrder.nativeOrder()).asIntBuffer();
    int buffh = Sys.getDirectBufferAddress(buff);
    Canvas.gl.readPixels(0,0,Display.getWidth(), Display.getHeight(), Canvas.gl.BGRA, Canvas.gl.UNSIGNED_BYTE, buffh);
    // create image
    BufferedImage img = new BufferedImage(Display.getWidth(), Display.getHeight(), BufferedImage.TYPE_INT_RGB);
    for (int ix=0;ix<Display.getWidth();ix++){
        for (int iy=0;iy<Display.getHeight();iy++){
            img.setRGB(ix, iy, buff.get((Display.getHeight()-iy-1)*Display.getWidth()+ix));
        }
    }
    // save file
    try {
        File out = new File (filename);
        ImageIO.write(img, "png", out);
    } catch (Exception e){
        e.printStackTrace();
    }
}

/* ---------------- */
// updateDistanceMaps()
/* ---------------- */
private void updateDistanceMaps() {
    // should it check the gMode ?????

    // updating can be reduced to every ?10 frames or so
    // or better can be split into first half, second half etc
    // or even easier only ?3 simthings are updated at a time

    // local variables
    Vector2f tVector;
    ArrayList tArrayA = new ArrayList();
    ArrayList tArrayB = new ArrayList();

//     for performance reason AND style issues
//     we only check the map every once in a while
//     mySimthing = + 00
//     aSimthing  = + 10
//     cursor?    = + 20

if (localTime % 30 == 0) {
    /* ---------------- */
    // mySimthing
    /* ---------------- */
    if (pMySimthing != null) {
        /* -------- */
        // dumbObject
        /* -------- */
        tArrayA.clear();
        tArrayB.clear();
        tVector = new Vector2f();
        pMySimthing.testArea = 10.0f;
        DumbObject tDumbObject;
        // find objects in the area
        for (int i = 0; i < dumbObjectArray.size(); i++) {
            tDumbObject = ( (DumbObject) dumbObjectArray.get(i));
            tDumbObject.hoppelFlag = false;
            if (Math.abs(tDumbObject.position.x - pMySimthing.position.x) <
                pMySimthing.testArea
                &&
                Math.abs(tDumbObject.position.y - pMySimthing.position.y) <
                pMySimthing.testArea) {
                tArrayA.add(tDumbObject);
            }
        }
        // find objects in FOV !!¬†very close
        for (int i = 0; i < tArrayA.size(); i++) {
            tDumbObject = ( (DumbObject) tArrayA.get(i));
            tVector.sub(tDumbObject.position, pMySimthing.position);
            if (Math.toDegrees(tVector.angle(pMySimthing.velocity)) <
                pMySimthing.FOV && pMySimthing.velocity.length() != 0.0f) {
                tArrayB.add(tDumbObject);
            }
            else {
                // nothing
            }
        }
        // find closest object
        if (tArrayB.size() >= 1) {
            DumbObject nearDumbObject = (DumbObject) tArrayB.get(0);
            tVector.sub(pMySimthing.position, nearDumbObject.position);
            float distanceToDumbObject = tVector.length();
            float tDistance = 0;
            for (int i = 1; i < tArrayB.size(); i++) {
                tDumbObject = ( (DumbObject) tArrayB.get(i));
                tVector.sub(pMySimthing.position, tDumbObject.position);
                tDistance = tVector.length();
                if (tDistance < distanceToDumbObject) {
                    distanceToDumbObject = tDistance;
                    nearDumbObject = tDumbObject;
                }
            }
            pMySimthing.nearDumbObject = nearDumbObject;
            pMySimthing.distanceToDumbObject = distanceToDumbObject;
            nearDumbObject.hoppelFlag = true;
        }
        else {
            pMySimthing.nearDumbObject = null;
            pMySimthing.distanceToDumbObject = 0;
        }
        // -> nearDumbObject
        // -> distanceToDumbObject

        /* -------- */
        // aSimthing
        /* -------- */
        tArrayA.clear();
        tArrayB.clear();
        tVector = new Vector2f();
        ASimthing tASimthing;
        //
        for (int i = 0; i < aSimthingArray.size(); i++) {
            tASimthing = ( (ASimthing) aSimthingArray.get(i));
            if (Math.abs(tASimthing.position.x - pMySimthing.position.x) <
                pMySimthing.testArea
                &&
                Math.abs(tASimthing.position.y - pMySimthing.position.y) <
                pMySimthing.testArea) {
                tArrayA.add(tASimthing);
            }
        }
        // find objects in FOV
        for (int i = 0; i < tArrayA.size(); i++) {
            tASimthing = ( (ASimthing) tArrayA.get(i));
            tVector.sub(tASimthing.position, pMySimthing.position);
            if (Math.toDegrees(tVector.angle(pMySimthing.velocity)) <
                pMySimthing.FOV
                && pMySimthing.velocity.length() != 0.0f) {
                tArrayB.add(tASimthing);
            }
            else {
                // nothing
            }
        }
        // find closest object
        if (tArrayB.size() >= 1) {
            ASimthing nearASimthing = (ASimthing) tArrayB.get(0);
            tVector.sub(pMySimthing.position, nearASimthing.position);
            float distanceToASimthing = tVector.length();
            float tDistance = 0;
            for (int i = 1; i < tArrayB.size(); i++) {
                tASimthing = ( (ASimthing) tArrayB.get(i));
                tVector.sub(pMySimthing.position, tASimthing.position);
                tDistance = tVector.length();
                if (tDistance < distanceToASimthing) {
                    distanceToASimthing = tDistance;
                    nearASimthing = tASimthing;
                }
            }
            pMySimthing.nearASimthing = nearASimthing;
            pMySimthing.distanceToASimthing = distanceToASimthing;
        }
        else {
            pMySimthing.nearASimthing = null;
            pMySimthing.distanceToASimthing = 0;
        }
        // -> nearASimthing
        // -> distanceToASimthing
    }
}
```

```
else if ( (localTime + 10) % 30 == 0) {
    /* ---------------- */
    // aSimthing
    /* ---------------- */
    ASimthing myASimthing = null;
    for (int j = 0; j < aSimthingArray.size(); j++) {
        myASimthing = (ASimthing) aSimthingArray.get(j);
        if ( myASimthing != null && myASimthing.getLocalMode() != “challange” ) {
            // --------
            // dumbObject
            // --------
            tArrayA.clear();
            tArrayB.clear();
            tVector = new Vector2f();
            float testArea = 10.0f;
            float FOV = 15.0f;
            DumbObject tDumbObject;
            //
            for (int i = 0; i < dumbObjectArray.size(); i++) {
                tDumbObject = ( (DumbObject) dumbObjectArray.get(i));
                if (Math.abs(tDumbObject.position.x - myASimthing.position.x) <
                    testArea
                    &&
                    Math.abs(tDumbObject.position.y - myASimthing.position.y) <
                    testArea) {
                    tArrayA.add(tDumbObject);
                }
            }
            // find objects in FOV
            for (int i = 0; i < tArrayA.size(); i++) {
                tDumbObject = ( (DumbObject) tArrayA.get(i));
                tVector.sub(tDumbObject.position, myASimthing.position);
                if (Math.toDegrees(tVector.angle(myASimthing.velocity)) < FOV &&
                    myASimthing.velocity.length() != 0.0f) {
                    tArrayB.add(tDumbObject);
                }
            }
            // find closest object
            if (tArrayB.size() >= 1) {
                DumbObject nearDumbObject = (DumbObject) tArrayB.get(0);
                tVector.sub(myASimthing.position, nearDumbObject.position);
                float distanceToDumbObject = tVector.length();
                float tDistance = 0;
                for (int i = 1; i < tArrayB.size(); i++) {
                    tDumbObject = ( (DumbObject) tArrayB.get(i));
                    tVector.sub(myASimthing.position, tDumbObject.position);
                    tDistance = tVector.length();
                    if (tDistance < distanceToDumbObject) {
                        distanceToDumbObject = tDistance;
                        nearDumbObject = tDumbObject;
                    }
                }
                myASimthing.nearDumbObject = nearDumbObject;
                myASimthing.distanceToDumbObject = distanceToDumbObject;
            }
            else {
                myASimthing.nearDumbObject = null;
                myASimthing.distanceToDumbObject = 0;
            }
            // -> nearDumbObject
            // -> distanceToDumbObject

            /* -------- */
            // aSimthing
            /* -------- */
            tArrayA.clear();
            tArrayB.clear();
            tVector = new Vector2f();
            ASimthing tASimthing;
            //
            for (int i = 0; i < aSimthingArray.size(); i++) {
                tASimthing = ( (ASimthing) aSimthingArray.get(i));
                if (   Math.abs(tASimthing.position.x - myASimthing.position.x) < myASimthing.testArea
                    && Math.abs(tASimthing.position.y - myASimthing.position.y) < myASimthing.testArea) {
                    //     don t check yourelf
                    if (tASimthing != myASimthing) {
                        tArrayA.add(tASimthing);
                    }
                }
            }
            //     get current socialPower
            myASimthing.socialPower = 0;
            for (int i = 0; i < tArrayA.size(); i++) {
                tASimthing = ( (ASimthing)tArrayA.get(i) );
                for (int k = 0; k < myASimthing.relation.length; k++){
                    if ( tASimthing.IDNumber.equals(myASimthing.relation[k].IDNumber) ){
                        myASimthing.socialPower += myASimthing.relation[k].numberOfFriends;
                    }
                }
            }
            //     find objects in FOV
            for (int i = 0; i < tArrayA.size(); i++) {
                tASimthing = ( (ASimthing)tArrayA.get(i) );
                tVector.sub(tASimthing.position, myASimthing.position);
                if (Math.toDegrees(tVector.angle(myASimthing.velocity)) <
                    myASimthing.FOV && myASimthing.velocity.length() != 0.0f) {
                    tArrayB.add(tASimthing);
                }
            }
            //     find closest object
            if (tArrayB.size() >= 1) {
                ASimthing nearASimthing = (ASimthing) tArrayB.get(0);
                tVector.sub(myASimthing.position, nearASimthing.position);
                float distanceToASimthing = tVector.length();
                float tDistance = 0;
                for (int i = 1; i < tArrayB.size(); i++) {
                    tASimthing = ( (ASimthing) tArrayB.get(i));
                    tVector.sub(myASimthing.position, tASimthing.position);
                    tDistance = tVector.length();
                    if (tDistance < distanceToASimthing) {
                        distanceToASimthing = tDistance;
                        nearASimthing = tASimthing;
                    }
                }
                myASimthing.nearASimthing = nearASimthing;
                myASimthing.distanceToASimthing = distanceToASimthing;
            }
            else {
                myASimthing.nearASimthing = null;
                myASimthing.distanceToASimthing = 0;
            }
            // -> nearASimthing
            // -> distaceToASimthing
            // -> nearDumbObject
            // -> distaceToDumbObject
        }
    }

    /* -------------------------------- */
    // gMode methods
    /* -------------------------------- */
    public void setGMode(String newMode) {
        // they need to be updated according to the list ‘OBJEKTHIERARCHIE’
        // all modes in all subobjects must be changed!
        if (newMode == gMode) {
            if(Settings.DEBUG) System.out.println(“ERROR! gMode not changed”);
        }
        else if (newMode == “addUser”) {
            // simthingMain
            gMode = newMode;
            // UI
            UIMode = “addUser”;
            if(Settings.DEBUG) System.out.println(“gMode to addUser”);
        }
        else if (newMode == “exploration”) {
            // simthingMain
            gMode = newMode;
            // UI
            UIMode = “idle”;
            // aSimthing
            for (int i = 0; i < aSimthingArray.size(); i++) {
                ( (ASimthing) aSimthingArray.get(i)).setLocalMode(“wander”);
                //aSimthingArray.get(i).myMode = “wander”;
            }
            if(Settings.DEBUG) System.out.println(“gMode to exploration”);
        }
        else if (newMode == “simulation”) {
            // simthingMain
            gMode = newMode;
            // UI
            UIMode = “spectator”;
            // aSimthing
```

```
        for (int i = 0; i < aSimthingArray.size(); i++) {
            ( (ASimthing) aSimthingArray.get(i)).setLocalMode(“wander”);
        }
        if(Settings.DEBUG) System.out.println(“gMode to simulation”);
    }
    else {
        if(Settings.DEBUG) System.out.println(“*** Error in simthingMain.setGMode()”);
    }
}

public String getGMode() {
    return gMode;
}

/* --------------------------------- */
// transition
/* --------------------------------- */
public void setTransitionMode( int tDuration,
                               boolean tResetFlag,
                               String tTransitionMode,
                               String tReturnGMode) {
    gMode = “transition”;
    pTransition = new transitionSetting( tDuration,
                                         tResetFlag,
                                         tTransitionMode,
                                         tReturnGMode);
}

/* --------------------------------- */
// cycle
/* --------------------------------- */
public void cycle() {

    // check UI
    UI();

    // tick tick tick
    localTime++;

// updateDistanceMaps
updateDistanceMaps();

// check gModes

/* ---------------- */
// transition //
/* ---------------- */
if (gMode == “transition”) { // main.transition

    // map of all possible transitions -> make map ‘MODETRANSITIONS’
    //
    // (request)    simulation_(addUser) > enterNumber + enterPIN + addUser
    // (cancel)     addUser_simulation
    // (success)    addUser_exploration
    // (addUser)    badData_simulation
    // (done) exploration_simulation

    // check transitionMode
    if (pTransition.transitionMode == “simulation_enterNumber”) { // main.transition.simulation_enterNumber
        /* ---------------- */
        // just for testing
        /* ---------------- */
        if (Settings.NOREADER) {
            setGMode(“addUser”);
        }

        /* ---------------- */
        // simulation_enterNumber
        // user wants to add new simthing
        // does some intro stuff
        /* ---------------- */
        if (pTransition.transitionTime < pTransition.duration) {

            //     reset member variables for new user
            for (int i=0; i<myIDNumber.length; i++){
                myIDNumber[i] = ‘0’;
            }
            myIDNumber[1] = ‘1’;
            for (int i=0; i<myPIN.length; i++){
                myPIN[i] = ‘0’;
            }
            enterPointer = 2;

        // draw texture enter phone number
        if (Settings.GUI_CENTERED)
            Canvas.glOverlayAddUser(0, 0, 9, 512);
        else
            Canvas.glOverlayAddUser(256 - Settings.SCREEN[0] / 2, 0, 9, 512);

        // move camera just for looks
        Canvas.camera.MoveCamera((float)Math.cos(Math.toRadians(pTransition.transitionTime))*0.1f);

        // next step
        pTransition.transitionTime++;
    }
    else {
        // done!
        setTransitionMode(25*60*2, true, “enterNumber_enterPIN”, “”); // give them 2 min to enter number
    }
}

else if (pTransition.transitionMode == “enterNumber_enterPIN”) { // main.transition.enterNumber_enterPIN

    // ARE ALL MOBILPHONE NUMBERS 11 CHARACTERS LONG ?

    /* ---------------- */
    // enterNumber_enterPIN
    // we need myIDNumber
    /* ---------------- */
    if (pTransition.transitionTime < pTransition.duration) {
        // controller gfx
        Canvas.glOverlayAddUser(Settings.SCREEN[0] / 2 - 128 - Settings.GUI_CONTROLLEROFFSET,
                                128 - Settings.SCREEN[1] / 2 + Settings.GUI_CONTROLLEROFFSET,
                                0,
                                256);

        // draw texture enter phone number
        if (Settings.GUI_CENTERED)
            Canvas.glOverlayAddUser( 0, 0, 5, 512);
        else
            Canvas.glOverlayAddUser( 256-Settings.SCREEN[0]/2, 0, 5, 512);
        //     print myIDNumber on the screen
        if (Settings.GUI_CENTERED)
            Canvas.glOverlayPrint(myIDNumber.length * -8, -32,
                                  new String(myIDNumber), 1);
        else
            Canvas.glOverlayPrint(256-Settings.SCREEN[0]/2 + myIDNumber.length * -8, -32,
                                  new String(myIDNumber), 1);
        // cursor
        if (Settings.GUI_CENTERED)
            Canvas.glOverlayPrint(11 * -8 + enterPointer * 16, -34, “_”, 1);
        else
            Canvas.glOverlayPrint(256-Settings.SCREEN[0]/2 + 11 * -8 + enterPointer * 16, -34, “_”, 1);
        //     check for button_pressed
        if (BUTTON_01) {
            //     (+)
            int number = myIDNumber[enterPointer];
            number++;
            // 48 = ‘0’
            // 57 = ‘9’
            //     wrap around
            if (number > 57)
                number = 48;
            myIDNumber[enterPointer] = (char) number;
        }
        else if (BUTTON_02) {
            //       reset number
            myIDNumber[enterPointer] = (char) ‘0’;
        }
        else if (RIGHT) {
            //        01(0) next value
            if (enterPointer == myIDNumber.length - 1) {
                //     finish input
                setTransitionMode(25 * 60 * 5, true,
                                  “enterPIN_addUser”, “”);
                enterPointer = 0;
            }
            else {
                //     next number
                enterPointer++;
            }
        }
        else if (LEFT) {
            //        02(X) previous value
            if (enterPointer == 0) {
```

```
                    //     finish input
                    setTransitionMode(100, true, “userQuit_simulation”,
                                      “simulation”);
                    enterPointer = 0;
                }
                else {
                    //     previous number
                    enterPointer--;
                }
            } // if BUTTON
              //} // if every 10 steps
              // next step
            if (FORWARD||BACKWARDS||LEFT||RIGHT||BUTTON_01||BUTTON_02)pTransition.transitionTime=0;
            pTransition.transitionTime++;
    }
        else {
            // TIME OUT MAYBE?
            if (Settings.DEBUG)
                System.out.println(“* enter number - TIMED OUT”);
            setGMode(“simulation”);
        } // if transitionTime
} // if transitionMode

    else if (pTransition.transitionMode == “enterPIN_addUser”) { // main.transition.enterPIN_addUser
        /* ---------------- */
        // enterPIN_addUser
        // we need myPIN
        // go to addUser when done
        /* ---------------- */
        if (pTransition.transitionTime < pTransition.duration) {

            // controller gfx
            Canvas.glOverlayAddUser(Settings.SCREEN[0] / 2 - 128 - Settings.GUI_CONTROLLEROFFSET,
                                    128 - Settings.SCREEN[1] / 2 + Settings.GUI_CONTROLLEROFFSET,
                                    0,
                                    256);

            // draw texture enter phone number
            if (Settings.GUI_CENTERED)
                Canvas.glOverlayAddUser(0, 0, 6, 512);
            else
                Canvas.glOverlayAddUser( 256-Settings.SCREEN[0]/2, 0, 6, 512);
            //     print myIDNumber on the screen
            if (Settings.GUI_CENTERED)
                Canvas.glOverlayPrint(myPIN.length * -8, -32, “****”, 1);
            else
                Canvas.glOverlayPrint(256 - Settings.SCREEN[0] / 2 +
                                      myPIN.length * -8, -32, “****”, 1);
            //  cursor
            if (Settings.GUI_CENTERED)
                Canvas.glOverlayPrint(4 * -8 + enterPointer * 16, -34, “_”, 1);
            else
                Canvas.glOverlayPrint(256-Settings.SCREEN[0]/2 + 4 * -8 + enterPointer * 16, -34, “_”, 1);
            //     check for button_pressed
            if (BUTTON_01) {
                //     (-)
                int number = myPIN[enterPointer];
                number++;
                // 48 = ‘0’
                // 57 = ‘9’
                //     wrap around
                if (number > 57)
                    number = 48;
                myPIN[enterPointer] = (char) number;
            }
            else if (BUTTON_02) {
                //       (+)
                myPIN[enterPointer] = (char) ‘0’;
            }
            else if (RIGHT) {
                //        01(O) next value
                if (enterPointer == myPIN.length - 1) {
                    if (Settings.DEBUG)
                        System.out.println(“* user data “ + myPIN +
                                           “ > “ + myIDNumber);
                    //     finish input
                    setGMode(“addUser”);
                    enterPointer = 10;
                }
                else {
                    //     next number
                    enterPointer++;
                }
            }
            else if (LEFT) {
                //        02(X) previous value
                if (enterPointer == 0) {
                    //     finish input
                    //     reset member variables for new user
                    for (int i = 0; i < myPIN.length; i++) {
                        myPIN[i] = ‘0’;
                    }
                    enterPointer = 10;
                    // give them 2 min to enter number
                    setTransitionMode(25 * 60 * 5, true,
                                      “enterNumber_enterPIN”, “”);
                }
                else {
                    //     previous number
                    enterPointer--;
                }
            } // if BUTTON
              //} // if every 10 steps
              // next step
            if (FORWARD||BACKWARDS||LEFT||RIGHT||BUTTON_01||BUTTON_02)pTransition.transitionTime=0;
            pTransition.transitionTime++;
        }
            else {
                // TIME OUT MAYBE?
                if (Settings.DEBUG)
                    System.out.println(“* enter PIN - TIMED OUT”);
                setGMode(“simulation”);
            } // if transitionTime
    }

        else if (pTransition.transitionMode == “addUser_simulation”) { // main.transition.addUser_simulation
            /* ---------------- */
            // addUser_simulation
            // user cancelled adding new simthing
            /* ---------------- */
            if (pTransition.transitionTime < pTransition.duration) {
                // move camera just for looks
                Canvas.camera.MoveCamera((float)Math.cos(Math.toRadians(pTransition.transitionTime))*0.1f);
                // next step
                pTransition.transitionTime++;
            }
            else {
                // jump back
                if (pTransition.resetFlag) {
                    setGMode(pTransition.returnGMode);
                }
                else {
                    gMode = pTransition.returnGMode;
                }
            }
        }

        else if (pTransition.transitionMode == “addUser_exploration”) { // main.transition.addUser_exploration
            /* ---------------- */
            // addUser_exploration
            // user added successfully data
            // and gets mySimthing
            /* ---------------- */
            if (pTransition.transitionTime < pTransition.duration) {

                // draw texture enter phone number
                if (Settings.GUI_CENTERED)
                    Canvas.glOverlayAddUser(0, 0, 4, 512);
                else
                    Canvas.glOverlayAddUser( 256-Settings.SCREEN[0]/2, 0, 4, 512);
                // move camera just for looks
                Canvas.camera.MoveCamera((float)Math.cos(Math.toRadians(pTransition.transitionTime))*0.1f);
                // next step
                pTransition.transitionTime++;
            }
            else {
                // jump back
                if (pTransition.resetFlag) {
                    setGMode(pTransition.returnGMode);
                }
                else {
                    gMode = pTransition.returnGMode;
                }
            }
        }

        else if (pTransition.transitionMode == “badData_simulation”) { // main.transition.badData_simulation
```

```
    /* ---------------- */
    // badData_simulation
    // user had bad data
    // go back to simulation
    /* ---------------- */
    if (pTransition.transitionTime < pTransition.duration) {
        // tell user about bad data
        // draw texture enter phone number
        if (Settings.GUI_CENTERED)
            Canvas.glOverlayAddUser(0, 0, 8, 512);
        else
            Canvas.glOverlayAddUser( 256-Settings.SCREEN[0]/2, 0, 8, 512);
        // next step
        pTransition.transitionTime++;
    }
    else {
        // change to addUser mode
        setGMode(pTransition.returnGMode);
    }
}

else if (pTransition.transitionMode == "userQuit_simulation") { // main.transition.badData_simulation
    /* ---------------- */
    // userQuit_simulation
    /* ---------------- */
    if (pTransition.transitionTime < pTransition.duration) {
        // tell user about bad data
        // draw texture enter phone number
        if (Settings.GUI_CENTERED)
            Canvas.glOverlayAddUser(0, 0, 10, 512);
        else
            Canvas.glOverlayAddUser( 256-Settings.SCREEN[0]/2, 0, 10, 512);
        // !-----------------------------------------------------^wrong number

        // next step
        pTransition.transitionTime++;
    }
    else {
        // change to addUser mode
        setGMode(pTransition.returnGMode);
    }
}


    else if (pTransition.transitionMode == "exploration_simulation") { // main.transition.exploration_simulation
        /* ---------------- */
        // exploration_simulation
        // user is done exploring world
        /* ---------------- */
        if (pTransition.transitionTime < pTransition.duration) {

            // draw texture enter phone number
            if (Settings.GUI_CENTERED)
                Canvas.glOverlayAddUser(0, 0, 3, 512);
            else
                Canvas.glOverlayAddUser( 256-Settings.SCREEN[0]/2, 0, 3, 512);
            // move camera just for looks
            Canvas.camera.MoveCamera((float)Math.cos(Math.toRadians(pTransition.transitionTime))*0.1f);
            // next step
            pTransition.transitionTime++;
        }
        else {
            //     jump back
            //     create simthing from collected data
            aSimthingArray.add(new ASimthing(
                                               canvas,
                                               this,
                                               aSimthingArray.size(),
                                               new String(myIDNumber),
                                               MySimthing.numbers));
            updateSimthingRelation();
            //     reset camera
            Canvas.resetViewport(50.0f + followCameraMinimumDistance);
            // reset idle Timer
            idleTime = 0;
            // set gMode
            setGMode(pTransition.returnGMode);
        }
    }
    else { // main.transition.error
        if(Settings.DEBUG) System.out.println("*** Error in main.transition.error");
    }
}
```

```
/* ---------------- */
// addUser    //
/* ---------------- */
else if (gMode == "addUser") { // main.addUser
                                // draw texture enter phone number
    if (Settings.GUI_CENTERED)
        Canvas.glOverlayAddUser(0, 0, 3, 512);
    else
        Canvas.glOverlayAddUser( 256-Settings.SCREEN[0]/2, 0, 3, 512);
    // create aSimthing
    String[] myUserData = null;
    if (Settings.NOREADER){
        //   FOOOOOOOOOOOOOOOOOOOOOOOOOOOO
        //   FAKE
        myUserData = new String[(int)(Math.random()*140.0)];
        myUserData[0] = "01794508664";
        myUserData[1] = "01739912110";
        myUserData[2] = "01793938833";
        myUserData[3] = "03066763322";
        myUserData[4] = "01793938835";
        myUserData[5] = "01793938836";
        for (int i=6; i<myUserData.length; i++){
            myUserData[i]="017"+(int)(Math.random()*100000000);
        }
        pMySimthing = new MySimthing(myUserData);
        setTransitionMode(100, true, "addUser_exploration", "exploration");
        // check if older version exists
        for (int i = 0; i < aSimthingArray.size(); i++) {
            String testForDoop = new String( ( (ASimthing)
                                                aSimthingArray.get(i)).IDNumber);
            if (testForDoop.equals(new String(myIDNumber))) {
                if (Settings.DEBUG)
                    System.out.println( "removed older version of " + testForDoop);
                aSimthingArray.remove(i);
                break;
            }
        }
        if (Settings.DEBUG)
            System.out.println("* creating FOOOOOOO mySimthing");
        Canvas.resetViewport(50);
        setTransitionMode(100, true, "addUser_exploration", "exploration");
    } else {
        SIMCardReader cardReader = new SIMCardReader();
        myUserData = cardReader.getData(new String(myPIN));
        // dump data
        if (Settings.DEBUG) {
            System.out.println("\n* carDataDump");
            for (int i = 0; i < myUserData.length; i++) {
                System.out.println(myUserData[i]);
            }
            System.out.println();
        }
        if (myUserData != null) {
            if (myUserData[0] == "error") {
                if (Settings.DEBUG)
                    System.out.println("* stopped creating mySimthing");
                setTransitionMode(100, true, "badData_simulation", "simulation");
            }
            else {
                // write data to HD
                DataWriter.write(new String(myIDNumber), myUserData);
                //     create a new mySimthing
                pMySimthing = new MySimthing(myUserData);
                //     check if number exists
                for (int i = 0; i < aSimthingArray.size(); i++) {
                    String testForDoop = new String( ( (ASimthing)
                                                        aSimthingArray.get(i)).IDNumber);
                    if (testForDoop.equals(new String(myIDNumber))) {
                        if (Settings.DEBUG)
                            System.out.println( "removed older version of " + testForDoop);
                        aSimthingArray.remove(i);
                        break;
                    }
                }
                if (Settings.DEBUG)
                    System.out.println("* creating mySimthing");
                Canvas.resetViewport(50);
                setTransitionMode(100, true, "addUser_exploration", "exploration");
            }
        } // if myUserData
    } // if NOREADER
```

```
        /* ---------------- */
        // update objects
        /* ---------------- */
        // aSimthing
        for (int i = 0; i < aSimthingArray.size(); i++) {
            ( (ASimthing) aSimthingArray.get(i)).cycle(gMode);
        }
        // dumbObject
        for (int i = 0; i < dumbObjectArray.size(); i++) {
            ( (DumbObject) dumbObjectArray.get(i)).drawMe();
        }

    }

    /* ---------------- */
    // exploration//
    /* ---------------- */
    else if (gMode == “exploration”) { // main.exploration
        /* ---------------- */
        // prepare view
        /* ---------------- */
        // controller gfx
        Canvas.glOverlayAddUser(Settings.SCREEN[0] / 2 - 128 - Settings.GUI_CONTROLLEROFFSET,
                                128 - Settings.SCREEN[1] / 2 + Settings.GUI_CONTROLLEROFFSET,
                                1,
                                256);
        // camera
        Vector2f vectorBetweenCursorAndCamera = new Vector2f(pMySimthing.
                                                             cursorPosition.x - Canvas.camera.m_vPosition.x,
                                                             pMySimthing.cursorPosition.y - Canvas.camera.m_vPosition.z);
        float maximumDistance = 25.0f;
        float minimumDistance = 22.0f;
        float actualDistance = vectorBetweenCursorAndCamera.length();
        float speed = 0.03f;
        if (actualDistance > maximumDistance) {
            Canvas.camera.MoveCamera( (actualDistance - maximumDistance) * speed);
        }
        if (actualDistance < minimumDistance) {
            Canvas.camera.MoveCamera( (actualDistance - minimumDistance) * speed);
        }

        Canvas.camera.PointCamera(pMySimthing.cursorPosition.x, 0.0f,
                                  pMySimthing.cursorPosition.y);

        /* ---------------- */
        // update objects
        /* ---------------- */
        // aSimthing
        for (int i = 0; i < aSimthingArray.size(); i++) {
            ( (ASimthing) aSimthingArray.get(i)).cycle(gMode);
        }
        // mySimthing
        pMySimthing.cycle(gMode);

    }

    /* ---------------- */
    // simulation//
    /* ---------------- */
    else if (gMode == “simulation”) { // main.simulation

        // controller gfx
        Canvas.glOverlayAddUser(Settings.SCREEN[0] / 2 - 128 - Settings.GUI_CONTROLLEROFFSET,
                                128 - Settings.SCREEN[1] / 2 + Settings.GUI_CONTROLLEROFFSET,
                                2,
                                256);

        // ---------------- //
        // follow camera
        // ---------------- //
        float followCameraSpeed = 0.06f;
        // choose ASimthing to follow
        ASimthing followASimthing = (ASimthing) aSimthingArray.get(followASimthingIndex);
        if (followASimthing != null) {
            Vector2f followCameraVelocity = new Vector2f();
            followCameraVelocity.sub(followASimthing.position,
                                     followCameraPosition);
            followCameraVelocity.scale(followCameraSpeed);
            followCameraPosition.add(followCameraVelocity);
        }

        Vector2f vectorBetweenFollowCameraAndCamera = new Vector2f(followCameraPosition.x - Canvas.camera.m_vPosition.x,
                                                                   followCameraPosition.y - Canvas.camera.m_vPosition.z);
        //     make distance to observed object adjustable
        float minimumDistance = followCameraMinimumDistance;
        float maximumDistance = followCameraMinimumDistance + 3.0f; // <- shall this be more flexible
        float actualDistance = vectorBetweenFollowCameraAndCamera.length();
        float speed = 0.01f; // !!!!
        if (actualDistance > maximumDistance) {
            Canvas.camera.MoveCamera( (actualDistance - maximumDistance) *
                                      speed);
        }
        if (actualDistance < minimumDistance) {
            Canvas.camera.MoveCamera( (actualDistance - minimumDistance) *
                                      speed);
        }

        Canvas.camera.PointCamera(followCameraPosition.x, 0.0f,
                                  followCameraPosition.y);

        /* ---------------- */
        // update objects
        /* ---------------- */
        // aSimthing
        for (int i = 0; i < aSimthingArray.size(); i++) {
            ( (ASimthing) aSimthingArray.get(i)).cycle(gMode);
        }
        // dumbObject
        for (int i = 0; i < dumbObjectArray.size(); i++) {
            ( (DumbObject) dumbObjectArray.get(i)).drawMe();
        }

    }
    else { // main.error
        if(Settings.DEBUG) System.out.println(“*** Error in main.error”);
    }

}

/* ---------------------------------- */
// UI
// to check user interaction ( UI )
/* ---------------------------------- */
public void UI() {

    /* ---------------- */
    // check idle time
    /* ---------------- */
    if (gMode == “simulation”) {
        if (! (BUTTON_01 || BUTTON_02 || FORWARD || BACKWARDS || LEFT ||
               RIGHT)) {
            idleTime++;
            if (idleTime > 500) {
                idleTime = 0;
                if (Settings.DEBUG)
                    System.out.println(“* IDLE choosing new perspektive”);
                //     choose random aSimthing
                followASimthingIndex = (int) (Math.random() *
                                              aSimthingArray.size());
                //     choose random distance
                followCameraMinimumDistance +=
                    (float) (Math.random() * 5.0f - 2.5f);
                if (followCameraMinimumDistance >= 50)
                    followCameraMinimumDistance = 50.0f;
                if (followCameraMinimumDistance <= 0)
                    followCameraMinimumDistance = 0.1f;
            }
        }
        else {
            idleTime = 0;
        }
    }

    /* ---------------- */
    // transition //
    /* ---------------- */
    if (gMode == “transition”) { // UI.transition
        if (BUTTON_02) {
            // handled in transition
        }
    }
```

```
/* ---------------- */
// exploration//
/* ---------------- */
else if (gMode == “exploration”) { // UI.exploration
    /* ---------------- */
    // idle //
    /* ---------------- */
    if (UIMode == “idle”) { // UI.exploration.idle
        /* ---------------- */
        // check keys //
        /* ---------------- */
        if (ROTATE_LEFT || ROTATE_RIGHT || FORWARD || BACKWARDS || LEFT ||
            RIGHT) {
            /* ---------------- */
            // STEERING  etc.   //
            /* ---------------- */
            Vector2f vectorBetweenCursorAndCamera = new Vector2f(pMySimthing.
                                                                 cursorPosition.x - Canvas.camera.m_vPosition.x,
                                                                 pMySimthing.cursorPosition.y - Canvas.camera.m_vPosition.z);
            Vector2f sideVector = new Vector2f();
            float speed = 0.2f;
            vectorBetweenCursorAndCamera.normalize();
            vectorBetweenCursorAndCamera.scale(speed);
            sideVector.cross(vectorBetweenCursorAndCamera);
            if (FORWARD) {
                pMySimthing.cursorPosition.add(vectorBetweenCursorAndCamera);
            }
            if (BACKWARDS) {
                pMySimthing.cursorPosition.sub(vectorBetweenCursorAndCamera);
            }
            if (LEFT) {
                pMySimthing.cursorPosition.sub(sideVector);
            }
            if (RIGHT) {
                pMySimthing.cursorPosition.add(sideVector);
            }
        }
        if (BUTTON_01) {
            /* ---------------- */
            // CONTACT    //
            /* ---------------- */
        }
        if (BUTTON_02) {
            /* ---------------- */
            // QUIT EXPLORATION //
            /* ---------------- */
            setTransitionMode(200, true, “exploration_simulation”, “simulation”);
        }
        // ( NEXT CYCLE )
    }
    /* ---------------- */
    // encounter  //
    /* ---------------- */
    else if (UIMode == “encounter”) { // UI.exploration.encounter
        if (true) {
            UIMode = “idle”;
        }
        else {
            // ( NEXT CYCLE )
        }
    }
    else {
        if(Settings.DEBUG) System.out.println(“*** Error in UI.exploration.error”); // UI.exploration.error
    }
}
/* ---------------- */
// simulation //
/* ---------------- */
else if (gMode == “simulation”) { // UI.simulation
    if (UIMode == “spectator”) { // UI.simulation.spectator
        /* ---------------- */
        // spectator  //
        /* ---------------- */
        if (ROTATE_LEFT || ROTATE_RIGHT || FORWARD || BACKWARDS || LEFT ||
            RIGHT) {
            /* ---------------- */
            // camera movement  //
            /* ---------------- */
            if (LEFT) {
                //Canvas.camera.StrafeCamera( -1.0f);
                followASimthingIndex--;
                if (followASimthingIndex == -1) followASimthingIndex=aSimthingArray.size()-1;
                if (Settings.DEBUG)System.out.println(“camera follows “+
                                                      ((ASimthing)aSimthingArray.get(followASimthingIndex)).IDNumber);
            }
            if (RIGHT) {
                //Canvas.camera.StrafeCamera(1.0f);
                followASimthingIndex++;
                if (followASimthingIndex == aSimthingArray.size()) followASimthingIndex=0;
                if (Settings.DEBUG)System.out.println(“camera follows “+
                                                      ((ASimthing)aSimthingArray.get(followASimthingIndex)).IDNumber);
            }
            if (FORWARD) {
                followCameraMinimumDistance -= 1.0f;
                if (followCameraMinimumDistance <= 0.0f)
                    followCameraMinimumDistance = 0.1f;
                if (Settings.DEBUG)
                    System.out.println(“followCameraMinimumDistance “ +
                                       followCameraMinimumDistance);
            }
            if (BACKWARDS) {
                followCameraMinimumDistance += 1.0f;
                if (followCameraMinimumDistance >= 50.0f)
                    followCameraMinimumDistance = 50.0f;
                if (Settings.DEBUG)
                    System.out.println(“followCameraMinimumDistance “ +
                                       followCameraMinimumDistance);
            }
        }
        else if (BUTTON_01) {
            /* ---------------- */
            // request new user //
            /* ---------------- */
            setTransitionMode(400, true, “simulation_enterNumber”, “addUser”);
        }
    }
    else {
        if(Settings.DEBUG) System.out.println(“*** Error in UI.simulation.error”); // UI.simulation.error
    }
}

/* ---------------- */
// addUser    //
/* ---------------- */
else if (gMode == “addUser”) { // UI.addUser
    /* ---------------- */
    // ABORD//
    /* ---------------- */
    if (BUTTON_02) {
        // QUIT
        // handled in transitions
    }
}
else {
    if(Settings.DEBUG) System.out.println(“*** Error in UI.error”); // UI.error
}
}

/* ---------------------------------------------------------------- */
// member classes
/* ---------------------------------------------------------------- */

/* -------------------------------- */
// transitionSetting ( structure )
/* -------------------------------- */

private class transitionSetting {

    int duration;
    int transitionTime;
    boolean resetFlag;
    String transitionMode;
    String returnGMode;
    String formerGMode;

    transitionSetting(int tDuration, boolean tResetFlag, String tTransitionMode,
                      String tReturnGMode) {
        duration = tDuration;
        transitionTime = 0;
        resetFlag = tResetFlag;
        transitionMode = tTransitionMode;
        returnGMode = tReturnGMode;
        formerGMode = gMode;
    }
}
}
```

ASIMTHING *ASimthing ist die Klasse für die Steuerung und Darstellung der autonomen Objekte zuständig ist. Alle Regeln für das Verhalten der Objekte untereinander sind hier festgelegt.*

```
//
//
// simthing.ASimthing
//
//

package simthing;

import org.lwjgl.*;
import org.lwjgl.opengl.*;
import org.lwjgl.input.*;

import java.lang.Math;
import java.util.*;

public class ASimthing {
    // ---------------------------------------------------------------- //
    // variables
    // ---------------------------------------------------------------- //
    public Canvas pCanvas;
    public SimthingMain pSimthingMain;

    public int ID;
    public String IDNumber;
    public String[] numbers;

    private String localMode = new String();
    public String gMode;

    public int localTime = 0;
    public int passingTime = 0;
    public int durationOfChallenge = 100;
    public int durationOfFleeing = 250;
    public int durationOfStandtall = 175;
    public int durationOfIdle = 180;

    private String DreamNumber = new String();

    public Relation pContact = null;

public float height = 5.0f;
public Vector2f position = new Vector2f((float)(Math.random()*Settings.TERRAIN_SIZE*2-Settings.TERRAIN_SIZE),
                                        (float)(Math.random()*Settings.TERRAIN_SIZE*2-Settings.TERRAIN_SIZE));
public Vector2f velocity = new Vector2f();
public float orientation = 0.0f;

// distance maps
public float FOV = 60.0f;
public float testArea = 5.0f;
public float minDistance = 1.0f;

// relation and data
public Relation[] relation;
public int socialPower; // is defined in update neighbors

// ---------------- //
// behavior specific
// ---------------- //
//     behavior
private Vector2f new_steering = new Vector2f();
//     behavior.wander
private Vector2f wander_velocity = new Vector2f();
private float steering_offset = 0.0f;
private float steering_change = 0.0006f;
private float steering_limit = 0.003f;
private float random_velocity_offset = 0.0f;
private float random_velocity_change = 0.001f;
private float random_velocity_limit = 0.005f;
//     behavior.avoidObject
private Vector2f avoidObject_velocity = new Vector2f();
//     peer
private float turnStyle;

// ---------------- //
// applySteering
// ---------------- //
private float max_force = 0.01f;
private float mass = 10.0f;
private float max_speed = 0.1f;
private float min_speed = 0.005f;

// ---------------- //
// other objects
// ---------------- //
//     dumbObject
public DumbObject nearDumbObject = null;
public float distanceToDumbObject = 0;
//     aSimthing
public ASimthing nearASimthing = null;
public float distanceToASimthing = 0;
//     mySimthing
public MySimthing nearMySimthing = null;
public float distanceToMySimthing = 0;

// ---------------------------------------------------------------- //
// constructors
// ---------------------------------------------------------------- //
public ASimthing(Canvas tPCanvas, SimthingMain tPSimthingMain, int tID,
                 String tIDNumber, String[] tNumbers) {
    // ---------------- //
    // objects
    // ---------------- //
    ID = tID;
    pCanvas = tPCanvas;
    pSimthingMain = tPSimthingMain;
    IDNumber = tIDNumber;
    numbers = tNumbers;

    // ---------------- //
    // behavior specific
    // ---------------- //
    if (Settings.RELATION_DEBUG)
        System.out.println("* simthing.simthingMain.aSimthing #" + ID +
                           " -> " + IDNumber);
    setLocalMode("wander");

}

// ---------------------------------------------------------------- //
// methods
// ---------------------------------------------------------------- //
// ---------------- //
// cycle()
// ---------------- //
public void cycle(String tGMode) {
    //     the current global mode
    gMode = tGMode;

    //     draw da info stuff
    if (Settings.DRAW_DEBUG)
        drawConnectLines();

    // ---------------- //
    // exploration      //
    // ---------------- //
    if (gMode == "exploration") { // aSimthing.exploration
        exploration();
    }
    // ---------------- //
    // simulation      //
    // ---------------- //
    else if (gMode == "simulation") {
        exploration(); // it s supposed to have it s own method / SPAGHETTI CODE
    }
    else {
        if (Settings.RELATION_DEBUG)
            System.out.println("*** Error in aSimthing.gMode > " + gMode);
    }
    localTime++;
}

// ---------------- //
// exploration      //
// ---------------- //
private void exploration() {

    //     check the local modes

    // ---------------- //
    // wander           //
    // ---------------- //
    if (localMode == "wander") {
        explorationWander();
        if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                             position.y, height)) {
            explorationWanderDraw();
        }
    }
```

```
// ---------------- //
// peer             //
// ---------------- //
else if (localMode == “peer”) { // aSimthing.exploration.peer
                                //
    explorationPeer();
    //     stick around deepending on numberOfNeighbors ???
    //     might change localMode to >>> wander
}
// ---------------- //
// challenge        //
// ---------------- //
else if (localMode == “challenge”) { // aSimthing.exploration.challenge
    explorationChallenge();
}
// ---------------- //
// flee             //
// ---------------- //
else if (localMode == “flee”) { // aSimthing.exploration.flee
    if (passingTime == localTime) {
        //     go back to wandering mode
        setLocalMode(“wander”);
    }
    else {
        //     perfom some looser action
        //     wander into the opposite direction of challenger
        setOrientation();
        new_steering.set(behavior_flee(pContact.pASimthing.position));
        applySteering(new_steering);
    }

    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                         position.y, height)) {
        //     get a smoothing sizin of the sphere
        //     get the current scale
        float currentTime = durationOfFleeing - (passingTime - localTime);
        double myScale = Math.pow((double)((2.0f * currentTime - (float)durationOfFleeing)/(float)durationOfFleeing), 6.0);
        drawObject( (float) Math.random() * 0.5f + 2.0f,
                    (float)(myScale*0.5+0.5),
                    (- myScale + 1.0)* 1.5,
                    false, true, true);
    }
}
// ---------------- //
// standtall        //
// ---------------- //
else if (localMode == “standtall”) { // aSimthing.exploration.standtall
    if (passingTime == localTime) {
        //     go back to wandering mode
        setLocalMode(“wander”);
    }
    //     perform some angeber action
    //     to show that you are the winner
    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                         position.y, height)) {
        explorationStandtallDraw();
    }
}
// ---------------- //
// meetfriend       //
// ---------------- //
else if (localMode == “meetfriend”) { // aSimthing.exploration.meetfriend

    // >>> jumping around and performing some crazy action
    if (Settings.RELATION_DEBUG)
        explorationMeetfriend();
}
// ---------------- //
// idle             //
// ---------------- //
else if (localMode == “idle”) { // aSimthing.exploration.idle
    if (passingTime == localTime) {
        //     remain idle or go wandering
        if (Math.random() < 0.7) {
            setLocalMode(“wander”);
        }
        else {
            if (Settings.RELATION_DEBUG)
                System.out.println(IDNumber + “ > zzzzZZZZ! (still) “);
            passingTime = localTime + durationOfIdle;
        }
    }
    //     perform some idle action
    //     somthing funny like exploding, falling, zzZZZZ!
        if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                             position.y, height)) {
            explorationIdleDraw();
        }
    }
    // ---------------- //
    // die              //
    // ---------------- //
    else if (localMode == “die”) { // aSimthing.exploration.die
                                   // if socialPower has been bad for some time you DIE! and be removed from list
        if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                             position.y, height)) {
            explorationDieDraw();
        }
    }
    else {
        if (Settings.RELATION_DEBUG)
            System.out.println(“*** Error in aSimthing.exploration > “ +
                               localMode);
    }
}

// ---------------- //
// setPhysicProperties
// ---------------- //
private void setPhysicProperties(String mode) {
    if (mode == “wander” || mode == “flee”) {
        //     behavior.wander
        wander_velocity = new Vector2f();
        steering_offset = 0.0f;
        steering_change = 0.001f;
        steering_limit = 0.003f;
        random_velocity_offset = 0.0f;
        random_velocity_change = 0.01f;
        random_velocity_limit = 0.03f;
        //     applySteering
        max_force = 0.5f;
        mass = 5.0f;
        max_speed = 0.1f;
        min_speed = 0.05f;
    }
    else if (mode == “peer”) {
        //     behavior.wander
        wander_velocity = new Vector2f();
        steering_offset = 0.0f;
        steering_change = 0.001f;
        steering_limit = 0.003f;
        random_velocity_offset = 0.0f;
        random_velocity_change = 0.01f;
        random_velocity_limit = 0.03f;
        //     applySteering
        max_force = 0.5f;
        mass = 10.0f;
        max_speed = 0.02f;
        min_speed = 0.01f;
    } else if (mode == “meetfriend”){
        //     behavior.wander
        wander_velocity = new Vector2f();
        steering_offset = 0.0f;
        steering_change = 0.001f;
        steering_limit = 0.003f;
        random_velocity_offset = 0.0f;
        random_velocity_change = 0.01f;
        random_velocity_limit = 0.03f;
        //     applySteering
        max_force = 0.5f;
        mass = 10.0f;
        max_speed = 0.02f;
        min_speed = 0.01f;
    }
}

// ---------------- //
// explorationMeetfriend
// ---------------- //
private void explorationMeetfriend() {
    //     if noone s around, just wander
    if (nearDumbObject == null && nearASimthing == null && nearMySimthing == null) {
        //     lost friend!
        if (passingTime < localTime) {
            passingTime = localTime + 90;
            turnStyle = (int)Math.random() * 2 - 1;
        }
        else {
```

```
        if (localTime == passingTime) {
            //     if there is still no one there
            //     after some cycles go to wandering mode
            setLocalMode(“wander”);
            passingTime = 0;
        }
        else {
            // !BRUTE FORCE actually needs to be done only once :)
            //     get vector to friend
            Vector2f vectorA = new Vector2f();
            vectorA.sub(pContact.pASimthing.position, position);
            //     get orientations
            float destinationOrientation = (float) Math.atan2( -vectorA.x, vectorA.y);
            float distanceTo = vectorA.length()*0.5f;

            //     look at friend
            //     calculate new orientation with turning speed
            orientation += (destinationOrientation - orientation) * 0.1;
        }
    }
    explorationMeetfriendDraw(true);
}
//     aSimthing
else if (nearASimthing != null &&
         ( (nearDumbObject == null && nearMySimthing == null)
           ||
           (distanceToASimthing < distanceToDumbObject &&
            distanceToASimthing < distanceToMySimthing)
           ||
           (nearDumbObject == null &&
            distanceToASimthing < distanceToMySimthing)
           ||
           (distanceToASimthing < distanceToDumbObject && nearMySimthing == null))) {
    //     what kind of aSimthing is close
    setContact(nearASimthing);
    //     is it a friend?
    if (pContact.knowEachOther == true) {
        explorationMeetfriendDraw(false);
    }
        //     is it a peer?
        else if (pContact.numberOfFriends > 0 || pContact.knowYou == true) {
            setLocalMode(“peer”);
            explorationMeetfriendDraw(false);
        }
        //     is it a fiend?
        else {
            setLocalMode(“wander”);
            explorationMeetfriendDraw(false);
        } //
    } // IF what kind of aSimthing
      //     mySimthing
    else if (nearMySimthing != null &&
             ( (nearDumbObject == null && nearASimthing == null)
               ||
               (distanceToMySimthing < distanceToDumbObject &&
                distanceToMySimthing < distanceToASimthing)
               ||
               (nearDumbObject == null &&
                distanceToMySimthing < distanceToASimthing)
               ||
               (distanceToMySimthing < distanceToDumbObject && nearASimthing == null))) {
        // draw object
        explorationMeetfriendDraw(true);
    }
    //     error
    else {
        if (Settings.RELATION_DEBUG)
            System.out.println(“error.ASimthing.exploration.wander”);
    }

}

// ---------------- //
// explorationChallenge
// ---------------- //
private void explorationChallenge() {
    // !BRUTE FORCE actually needs to be done only once :)
    //     get vector to challengePosition
    Vector2f vectorA = new Vector2f();
    vectorA.sub(pContact.pASimthing.position, position);
    //     get orientations
    float destinationOrientation = (float) Math.atan2( -vectorA.x, vectorA.y);
    float distanceTo = vectorA.length()*0.5f;
```

```
//     if time has passed...
if (passingTime == localTime) {
    //     check who is more @ home
    if (pContact.pASimthing.socialPower > socialPower) {
        // >>> flee
        if (Settings.RELATION_DEBUG)
            System.out.println(IDNumber + “ is fleeing!”+”\nME: “+socialPower+” / IT: “+pContact.pASimthing.socialPower);
        setLocalMode(“flee”);
        passingTime = localTime + durationOfFleeing;
    }
    else if (pContact.pASimthing.socialPower < socialPower) {
        // >>> standtall
        if (Settings.RELATION_DEBUG)
            System.out.println(IDNumber + “ is st...!”+”\nME: “+socialPower+” / IT: “+pContact.pASimthing.socialPower);
        setLocalMode(“standtall”);
        passingTime = localTime + durationOfStandtall;
    }
    else {
        // >>> draw! -> what should happen?
        // especially when both socialPowers are 0!
        // >>> flee
        if (Settings.RELATION_DEBUG)
            System.out.println(IDNumber + “ draw!”);
        setLocalMode(“flee”);
        passingTime = localTime + durationOfFleeing / 10;
    }
}
else {
    //     look at challenger
    //     calculate new orientation with turning speed
    orientation += (destinationOrientation - orientation) * 0.1;
}
// draw animated texture
if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2, position.y, height)) {
    drawObject( (float) Math.random() * 0.5f + 2.0f, 1.0f, 1.0,
                false, false, true);
    // draw scanner
    //     prepare drawing
    Canvas.gl.enable(GL.TEXTURE_2D);
    Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[ (localTime % 4) + 11]);
    Canvas.gl.enable(GL.BLEND);
    Canvas.gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
    Canvas.gl.color3f(1.0f, 1.0f, 1.0f);
    Canvas.gl.disable(GL.DEPTH_TEST);
    Canvas.gl.disable(GL.LIGHTING);
    if (Settings.FOG)
        Canvas.gl.enable(GL.FOG);
    else
        Canvas.gl.disable(GL.FOG);
    //     position and rotate koordinatensystem
    Canvas.gl.pushMatrix(); // scanner
    Canvas.gl.translatef(position.x, 0.0f, position.y);
    Canvas.gl.rotatef( 180.0f - (float) Math.toDegrees(orientation), 0.0f, 1.0f, 0.0f);
    Canvas.gl.rotatef( -90f, 1.0f, 0.0f, 0.0f);
    Canvas.gl.scalef(distanceTo, distanceTo, distanceTo);
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.popMatrix(); // scanner
  }
}

// ---------------- //
// explorationPeer
// ---------------- //
private void explorationPeer() {
    //     if noone s around, just wander
    if (nearDumbObject == null && nearASimthing == null && nearMySimthing == null) {
        //     lost peer!
        if (passingTime<localTime){
            passingTime = localTime + 90;//(int)Math.random()*90+45;
            turnStyle = (float)Math.random()*8.0f-4.0f;
        } else {
            if (localTime == passingTime){
                //     if there is still no one there
                //     after some cycles go to wandering mode
                setLocalMode(“wander”);
                passingTime = 0;
            } else {
                //     do some turning before ‘wander’
                orientation = (float)Math.toRadians(Math.toDegrees(orientation)+4.0);
                velocity.set((float)Math.sin(Math.toRadians(orientation)), (float)Math.cos(Math.toRadians(orientation)));
                velocity.scale(0.05f);
            }
        }
        explorationPeerDraw(true);
```

```
}
//     aSimthing
else if (nearASimthing != null &&
         ( (nearDumbObject == null && nearMySimthing == null)
           ||
           (distanceToASimthing < distanceToDumbObject &&
            distanceToASimthing < distanceToMySimthing)
           ||
           (nearDumbObject == null &&
            distanceToASimthing < distanceToMySimthing)
           ||
           (distanceToASimthing < distanceToDumbObject && nearMySimthing == null))) {
    //     what kind of aSimthing is close
    setContact(nearASimthing);
    //     is it a friend?
    if (pContact.knowEachOther == true) {
        if (Settings.RELATION_DEBUG)
            System.out.println(IDNumber + " > #" +
                               pContact.IDNumber +
                               " is a friend");
        setLocalMode("meetfriend");
        explorationPeerDraw(false);
    }
    //     is it still a peer?
    else if (pContact.numberOfFriends > 0 || pContact.knowYou == true) {
        //     calculates the new object orientation
        setOrientation();
        //     perform wandering action and save it to a vector
        new_steering.set(behavior_wander());
        //     apply new velocity - including speed restrains & world border restrains
        applySteering(new_steering);
        //     draw
        explorationPeerDraw(true);
    }
    //     is it a fiend?
    else {
        if (Settings.RELATION_DEBUG)
            System.out.println(IDNumber + " > #" + pContact.IDNumber +
                               " is a fiend");
            if (! (pContact.pASimthing.getLocalMode().equals("challenge"))
                && ! (pContact.pASimthing.getLocalMode().equals("flee"))) {
                //     pass my name to the other
                pContact.pASimthing.setContact(this);
                //     set other one to challenge
                pContact.pASimthing.setLocalMode("challenge");
                //     set myself to challenge
                setLocalMode("challenge");
                //     reset other passingTimer
                pContact.pASimthing.passingTime = pContact.pASimthing.localTime + pContact.pASimthing.durationOfChallenge;
                //     reset passingTimer
                passingTime = localTime + durationOfChallenge;
            }
            explorationPeerDraw(false);
        } //
    } // IF what kind of aSimthing
      //     mySimthing
    else if (nearMySimthing != null &&
             ( (nearDumbObject == null && nearASimthing == null)
               ||
               (distanceToMySimthing < distanceToDumbObject &&
                distanceToMySimthing < distanceToASimthing)
               ||
               (nearDumbObject == null &&
                distanceToMySimthing < distanceToASimthing)
               ||
               (distanceToMySimthing < distanceToDumbObject && nearASimthing == null))) {
        if (Settings.RELATION_DEBUG)
            System.out.println("mySimthing is around");
        new_steering.set(behavior_wander());
        //     apply new velocity
        applySteering(new_steering);
        // draw object
        explorationPeerDraw(true);
    }
    //     error
    else {
        if (Settings.RELATION_DEBUG)
            System.out.println("error.ASimthing.exploration.wander");
    }
}
```

```
// ---------------- //
// explorationWander//
// ---------------- //
private void explorationWander() {

    //     calculates the new object orientation
    setOrientation();

    // aSimthing.exploration.wander
    //     if noone s around, just wander
    if (nearDumbObject == null && nearASimthing == null && nearMySimthing == null) {
        //     perform wandering action and save it to a vector
        new_steering.set(behavior_wander());
        //     apply new velocity - including speed restrains & world border restrains
        applySteering(new_steering);
    }
    //     what is the closest object
    //     dumbObject
    else if (nearDumbObject != null &&
             ( (nearASimthing == null && nearMySimthing == null)
               ||
               (distanceToDumbObject < distanceToASimthing &&
                distanceToDumbObject < distanceToMySimthing)
               ||
               (nearASimthing == null &&
                distanceToDumbObject < distanceToMySimthing)
               ||
               (distanceToDumbObject < distanceToASimthing && nearMySimthing == null))) {
        if (localTime % 50 == 0) {
            avoidObject_velocity = behavior_avoidObject(nearDumbObject.
                                                        position);
        }
        new_steering.set(avoidObject_velocity);
        //     apply new velocity - including speed restrains & world border restrains
        applySteering(new_steering);
    }
    //     aSimthing
    else if (nearASimthing != null &&
             ( (nearDumbObject == null && nearMySimthing == null)
               ||
               (distanceToASimthing < distanceToDumbObject &&
                distanceToASimthing < distanceToMySimthing)
               ||
               (nearDumbObject == null &&
                distanceToASimthing < distanceToMySimthing)
               ||
               (distanceToASimthing < distanceToDumbObject && nearMySimthing == null))) {

        // what kind of aSimthing is close
        //     first find it in relation[]
        setContact(nearASimthing);

        if (pContact.knowEachOther == true) {
            if (Settings.RELATION_DEBUG)
                System.out.println(IDNumber + " > #" +
                                   pContact.IDNumber +
                                   " is a friend");
            //     friend
            setLocalMode("meetfriend");
        }
        else if (pContact.numberOfFriends > 0 || pContact.knowYou == true) {
            //     friendly ( this could also be an extra behavior like -> 'admire' or 'look up to' )
            //     same peer group
            if (Settings.RELATION_DEBUG)
                System.out.println(IDNumber + " > #" + pContact.IDNumber +
                                   " is a peer");
            setLocalMode("peer");
            passingTime = 0;
        }
        else {
            //     UNfriendly
            if (Settings.RELATION_DEBUG)
                System.out.println(IDNumber + " > #" + pContact.IDNumber +
                                   " is a fiend");
            if (! (pContact.pASimthing.getLocalMode().equals("challenge"))
                && ! (pContact.pASimthing.getLocalMode().equals("flee"))) {
                //     pass my name to the other
                pContact.pASimthing.setContact(this);
                //     set other one to challenge
                pContact.pASimthing.setLocalMode("challenge");
                //     set myself to challenge
                setLocalMode("challenge");
                //     reset other passingTimer
                pContact.pASimthing.passingTime = pContact.pASimthing.localTime + pContact.pASimthing.durationOfChallenge;
```

```
                //     reset passingTimer
                passingTime = localTime + durationOfChallenge;
            }
        }
    }

    //     mySimthing
    else if (nearMySimthing != null &&
             ( (nearDumbObject == null && nearASimthing == null)
               ||
               (distanceToMySimthing < distanceToDumbObject &&
                distanceToMySimthing < distanceToASimthing)
               ||
               (nearDumbObject == null &&
                distanceToMySimthing < distanceToASimthing)
               ||
               (distanceToMySimthing < distanceToDumbObject && nearASimthing == null))) {
        if (Settings.RELATION_DEBUG)
            System.out.println("mySimthing is around");
        new_steering.set(behavior_wander());
        //     apply new velocity
        applySteering(new_steering);
    }
    //     no one around
    else {
        if (Settings.RELATION_DEBUG)
            System.out.println("error.ASimthing.exploration.wander");
    }

    //     get borred by chance
    if (Math.random() > 0.9995) { // 0.999 is the best value / they can t be sleeping all the time
        if (Settings.RELATION_DEBUG)
            System.out.println(IDNumber + " > zzzzZZZZ!");
        setLocalMode("idle");
        passingTime = localTime + durationOfIdle;
    }
}

// ---------------- //
// applySteering
// ---------------- //
private void applySteering(Vector2f new_steering) {
    // supply vector objects
    Vector2f steering_direction = new Vector2f();
    Vector2f steering_force = new Vector2f();
    Vector2f acceleration = new Vector2f();

    // make borders unattractive
    Vector2f keepOnTerrain_velocity = new Vector2f();
    float terrainSensitivArea = 10.0f;
    if (position.x > Settings.TERRAIN_SIZE - terrainSensitivArea) {
        keepOnTerrain_velocity.x = ( (Settings.TERRAIN_SIZE - terrainSensitivArea) -
                                     position.x) / terrainSensitivArea;
    }
    if (position.x < terrainSensitivArea) {
        keepOnTerrain_velocity.x = (terrainSensitivArea - position.x) /
        terrainSensitivArea;
    }
    if (position.y > Settings.TERRAIN_SIZE - terrainSensitivArea) {
        keepOnTerrain_velocity.y = ( (Settings.TERRAIN_SIZE -
                                      terrainSensitivArea) -
                                     position.y) / terrainSensitivArea;
    }
    if (position.y < terrainSensitivArea) {
        keepOnTerrain_velocity.y = (terrainSensitivArea - position.y) /
        terrainSensitivArea;
    }
    keepOnTerrain_velocity.normalize();
    //     scale the border repell vector with a certain amount. does it make sense? i don t know yet
    keepOnTerrain_velocity.scale(0.01f);
    //     add it to the suggeseted...
    new_steering.add(keepOnTerrain_velocity);

    //     take the 'suggested' velocity modifikation as steering_direction
    steering_direction.set(new_steering);
    //     restrain it to the max_force
    steering_force.truncLength(max_force, steering_direction);
    //     apply the mass of the object
    acceleration.scale(1.0f / mass, steering_force);
    //     add it to the old velocity
    velocity.add(velocity, acceleration);
    //     restrain it to max_speed
    velocity.truncLength(max_speed);
    //     restrain it to min_speed
    //velocity.minLength( min_speed );
    //     add velocity to current position
    position.add(velocity);
}

// ---------------- //
// setLocalMode
// ---------------- //
public void setLocalMode(String new_mode) {
    if (new_mode == "wander" || new_mode == "peer" || new_mode == "meetfriend" || new_mode == "flee"){
        setPhysicProperties(new_mode);
    }
    localMode = new_mode;
}

// ---------------- //
// setContact
// ---------------- //
public void setContact(ASimthing contactASimthing) {
    for (int i = 0; i < relation.length; i++) {
        if (relation[i].IDNumber.equals(contactASimthing.IDNumber)) {
            pContact = relation[i];
            break;
        }
    }
}

// ---------------- //
// getLocalMode
// ---------------- //
public String getLocalMode() {
    return localMode;
}

// ---------------- //
// behavior.avoidObject
// ---------------- //
private Vector2f behavior_avoidObject(Vector2f avoidPosition) {
    //     supply vector objects
    Vector2f vectorA = new Vector2f();
    Vector2f vectorB = new Vector2f();
    //     get the normalized velocity
    vectorB.normalize(velocity);
    //     scale it to steering_limit
    vectorB.scale(steering_limit);
    //     get the vector between position and the obstacle position
    vectorA.sub(avoidPosition, position);
    vectorA.normalize();
    //     determine wheter to avoid obstacle to the left or to the right
    //     reset steering_offset so that the object continues smooth after yvoiding obstacle
    if (vectorA.angleReal(velocity) < 0) {
        //obstacle to the right
        vectorB.negate();
        steering_offset = -steering_limit;
    }
    else {
        //obstacle to the left
        steering_offset = steering_limit;
    }
    //     steer away from obstacle
    vectorA.scale( -steering_limit / 2.0f);
    //     steer 90∞ to own velocity
    vectorB.cross();
    //     add away vector to 90¬∞ vector
    vectorB.add(vectorA);
    //     return avoidObject_velocity
    return vectorB;
}

// ---------------- //
// behavior.flee
// ---------------- //
private Vector2f behavior_flee(Vector2f avoidPosition) {
    //     supply vector objects
    Vector2f vectorA = new Vector2f();
    //     get vector away from avoidposition
    vectorA.sub(position, avoidPosition);
    //     normlize vector
    vectorA.normalize();
    //     scale it to steering_limit
    vectorA.scale(max_speed * 2.0f);
    //     return flee_velocity
    return vectorA;
}
```

```
// ---------------- //
// behavior.wander
// ---------------- //
private Vector2f behavior_wander() {
    //     supply vector objects
    Vector2f steering_direction_wandering = new Vector2f();
    Vector2f vectorA = new Vector2f();
    //     local variables
    float turnSpeed;
    // don t let it get to slow -> ??? -> or else it might get borred :)
    velocity.minLength(min_speed);
    //     get the steering_direction from the old velocity
    steering_direction_wandering.cross(velocity);
    //     check if steering_direction_wandering is zero, we don t want that, because the whole wandering depends on movement
    if (steering_direction_wandering.lengthSquared() == 0) {
        steering_direction_wandering.y = 0.1f;
    }
    //     find out how fast the object can turn
    turnSpeed = (float) Math.pow(steering_direction_wandering.length() /
                                 max_speed, 0.1f);
    //     and normalize it
    steering_direction_wandering.normalize();
    //     add a random steering_change to steering_offset
    steering_offset += (float) Math.random() * steering_change -
        steering_change * 0.5f;
    //     and restrain to steering_limit
    steering_offset = Math.min(steering_limit, steering_offset);
    steering_offset = Math.max( -steering_limit, steering_offset);
    //     relate it to the steering_offset, so an object can t turn fast if it s not moving fast
    steering_offset *= turnSpeed;
    //System.out.println("turnSpeed                : "+turnSpeed);
    //     scale it with the steering direction
    steering_direction_wandering.scale(steering_offset);
    //     give the object the possibilit to slow down or speed up
    random_velocity_offset += (float) Math.random() *
        random_velocity_change -
        random_velocity_change * 0.5f;
    random_velocity_offset = Math.min(random_velocity_limit,
                                      random_velocity_offset);
    random_velocity_offset = Math.max( -random_velocity_limit,
                                      random_velocity_offset);
    //     take the normalized velocity
    vectorA.normalize(velocity);
    //     scale it with that random_velocity_offset to get a steering_offset
    vectorA.scale(random_velocity_offset);
    //     add steering_direction_wandering to velocity
    steering_direction_wandering.add(vectorA);
    //     include obstacle avoidance for aSimthing
    if (nearASimthing!=null){
        if (distanceToASimthing < 1.0f){
            //steering_direction_wandering.add(behavior_avoidObject(nearASimthing.position));
            Vector2f avoidPosition = new Vector2f();
            avoidPosition.sub(position, nearASimthing.position);
            avoidPosition.normalize();
            avoidPosition.scale(0.001f);
            steering_direction_wandering.add(avoidPosition);
        }
    }

    //     return the ergebnis
    return steering_direction_wandering;
}

// ---------------- //
// getLocalBase
// ---------------- //
private void getLocalBase() {
}

// ---------------- //
// setOrientation   //
// ---------------- //
private void setOrientation() {
    orientation = (float) Math.atan2( -velocity.x, velocity.y);
}

// ---------------- //
// simulationDraw   //
// ---------------- //
private void simulationDraw() {
    // frustumCullung
    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2, position.y, height)) {
        // fooObject
        drawObject(2.0f, 1.0f, 1.0, true, true, true);
    }
}

// ---------------- //
// explorationWanderDraw
// ---------------- //
private void explorationWanderDraw() {
    drawObject(2.0f, 1.0f, 1.0, false, true, true);
}

// ---------------- //
// explorationPeerDraw
// ---------------- //
private void explorationPeerDraw(boolean velocityPointer) {
    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2,
                                         position.y, height)) {
        drawObject(2.0f, 1.0f, 1.0, true, velocityPointer, true);
    }
}

// ---------------- //
// explorationStandtallDraw
// ---------------- //
private void explorationStandtallDraw() {
    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2, position.y, height)) {
        //     get the current scale
        float myScale;
        float currentTime = durationOfStandtall - ( passingTime - localTime );
        //     seperate scale in two parts, growing and shrinking
        float leftSide = 0.66f; // grow
        float rightSide = 0.33f; // shrink
        if (currentTime < leftSide*(float)durationOfStandtall) {
            myScale = (float)currentTime / (leftSide*(float)durationOfStandtall);
        } else {
            myScale = (float)(durationOfStandtall - currentTime)/ (rightSide*(float) durationOfStandtall);
        }
        drawObject(2.0f,
                   2.0f*myScale+1.0f,
                   1.0, true, false, true);
    }
}

// ---------------- //
// explorationMeetfriendDraw
// ---------------- //
private void explorationMeetfriendDraw(boolean velocityPointer) {
    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2, position.y,
                                         height)) {
        float hoppyPoppy = 60;
        float jumpModifier = - ( ( (localTime * 3.2f) % hoppyPoppy -
                                   hoppyPoppy / 2.0f) *
                                 ( (localTime * 3.2f) % hoppyPoppy -
                                   hoppyPoppy / 2.0f))/ ( (hoppyPoppy * hoppyPoppy) / 2);
        drawObject(jumpModifier + 1.0f,
                   1.0f,
                   1.0,
                   true,
                   velocityPointer,
                   true);
    }
}

// ---------------- //
// explorationIdleDraw
// ---------------- //
private void explorationIdleDraw() {
    //     smoothin
    float smoothin = 10.0f;
    float height;
    float currentTime = durationOfIdle - ( passingTime - localTime );
    if (currentTime < smoothin) {
        height = 2.0f - ( (2.0f-1.3f)/smoothin ) * currentTime; // from 2.0f to 1.3f
    } else if ( currentTime > durationOfIdle - smoothin) {
        height = 2.0f - ( (2.0f-1.3f)/smoothin ) * (durationOfIdle - currentTime); // from 1.3f to 2.0f
    }
    else {
        height = (float) Math.sin(Math.toRadians( ( (passingTime - localTime)
                                                    * (360 / (durationOfIdle-smoothin))) % 180)) * 0.3f + 1.0f;
    }
    drawObject(height,
               1.0f,
               1.0,
               false, false, false);
    // draw zzzZZ
```

```
//     prepare drawing
Canvas.gl.enable(GL.TEXTURE_2D);
Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[15]);
Canvas.gl.enable(GL.BLEND);
Canvas.gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
Canvas.gl.color3f(1.0f, 1.0f, 1.0f);
Canvas.gl.disable(GL.DEPTH_TEST);
Canvas.gl.disable(GL.LIGHTING);
if (Settings.FOG)
    Canvas.gl.enable(GL.FOG);
else
    Canvas.gl.disable(GL.FOG);
//     position and rotate koordinatensystem
Canvas.gl.pushMatrix(); // zzzZZ
Canvas.gl.translatef(position.x,
                     (float) Math.cos(Math.toRadians(((passingTime - localTime) * (360 / durationOfIdle)) % 90)) * 2.0f +
                     1.0f
                     , position.y);
Canvas.gl.rotatef( - (float) Math.toDegrees(orientation), 0.0f,
                  1.0f, 0.0f);
Canvas.gl.rotatef( -90f, 0.0f, 1.0f, 0.0f);
Canvas.gl.pushMatrix();
Canvas.gl.scalef(1.5f,1.5f,1.5f);
Canvas.gl.callList(Canvas.plane);
Canvas.gl.popMatrix();

// dream of friends
if (((passingTime - localTime) * (360 / durationOfIdle) % 90) == 0) {
    DreamNumber = numbers[(int)(Math.random()*numbers.length)];
    if(Settings.DEBUG)System.out.println(IDNumber + " > Dreaming of: "+DreamNumber);
}
// scale from small to big -> in a way not to be explained :)
float DreamNumberSize = 0.1f + 0.2f * ((100- (passingTime - localTime) * (360 / durationOfIdle) % 90)*0.01f);
Canvas.gl.pushMatrix(); // drawDreamNumber
Canvas.gl.translatef( 0.0f,
                      (float) Math.cos(Math.toRadians(((passingTime - localTime)
                      * (360 / durationOfIdle)) % 90)) * 1.5f + 0.5f,
                      0.0f);
Canvas.gl.scalef(DreamNumberSize, DreamNumberSize, DreamNumberSize);
    Canvas.gl3DPrint(DreamNumber, 1);
    Canvas.gl.popMatrix(); // drawDreamNumber
    Canvas.gl.popMatrix(); // zzzZZ
}

// ---------------- //
// explorationDieDraw
// ---------------- //
private void explorationDieDraw() {
    if (Canvas.g_Frustum.SphereInFrustum(position.x, height / 2, position.y, height)) {
        //drawFooObject();
    }
}

// ---------------- //
// drawObject
// ---------------- //
public void drawObject(float height, float scale, double sphereRadius,
                       boolean boundingSphere,
                       boolean velocityPointer,
                       boolean drawIDNumber) {
    //     draw object
    //     prepare drawing
    float sizeByNumberOfNumbers = (float) numbers.length / 150.0f + 0.5f; // size mod for numberOfNumbers
    Canvas.gl.enable(GL.TEXTURE_2D);
    Canvas.gl.enable(GL.BLEND);
    Canvas.gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
    Canvas.gl.color3f(1.0f, 1.0f, 1.0f);
    Canvas.gl.disable(GL.DEPTH_TEST);
    Canvas.gl.disable(GL.LIGHTING);
    if (Settings.FOG)
        Canvas.gl.enable(GL.FOG);
    else
        Canvas.gl.disable(GL.FOG);
    //     position and rotate koordinatensystem
    Canvas.gl.pushMatrix(); // main
    Canvas.gl.translatef(position.x, 0.0f, position.y);
    Canvas.gl.rotatef( - (float) Math.toDegrees(orientation), 0.0f, 1.0f, 0.0f);
    //     texture and draw shadow
    Canvas.gl.pushMatrix(); // schatten
    Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[4]);
    Canvas.gl.rotatef(90.0f, 1.0f, 0.0f, 0.0f);
    //     scale by behavior
    Canvas.gl.scalef(scale, scale, scale);
    //     scale by number of numbers
    Canvas.gl.scalef(sizeByNumberOfNumbers, sizeByNumberOfNumbers, sizeByNumberOfNumbers);
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.popMatrix(); // schatten
    Canvas.gl.pushMatrix(); // schattenSphere
    Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[7]);
    Canvas.gl.rotatef(90.0f, 1.0f, 0.0f, 0.0f);
    Canvas.gl.scalef( (float) sphereRadius*0.5f*scale,
                      (float) sphereRadius*0.5f*scale,
                      (float) sphereRadius*0.5f*scale);
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.popMatrix(); // schattenSphere
    //     texture and draw object
    Canvas.gl.pushMatrix(); // object
    Canvas.gl.translatef(0.0f, height, 0.0f);
    Canvas.gl.scalef(scale, scale, scale);
    //     sphere
    Canvas.gl.pushMatrix(); // sphere
    //     scale by number of numbers
    Canvas.gl.scalef((1+sizeByNumberOfNumbers)*0.5f, (1+sizeByNumberOfNumbers)*0.5f, (1+sizeByNumberOfNumbers)*0.5f);
    Canvas.gl.color3f(0.0f, 0.0f, 0.0f);
    Canvas.gl.disable(GL.TEXTURE_2D);
    Canvas.gl.disable(GL.BLEND);
    Canvas.gl.enable(GL.DEPTH_TEST);
    Canvas.glu.sphere(Canvas.sphere, sphereRadius*0.5, 8, 8);
    Canvas.gl.disable(GL.DEPTH_TEST);
    Canvas.gl.enable(GL.BLEND);
    Canvas.gl.enable(GL.TEXTURE_2D);
    Canvas.gl.popMatrix(); // sphere
    //     spikes
    //     scale by number of numbers
    Canvas.gl.scalef(sizeByNumberOfNumbers, sizeByNumberOfNumbers, sizeByNumberOfNumbers);
    Canvas.gl.color3f(1.0f, 1.0f, 1.0f);
    Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[3]);
    //     plane1
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.rotatef(90.0f, 1.0f, 0.0f, 0.0f);
    //     plane2
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.rotatef(90.0f, 0.0f, 1.0f, 0.0f);
    //     plane3
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.popMatrix(); // object
    //     texture and draw boundingSphere
    if (boundingSphere) {
        Canvas.gl.pushMatrix(); // boundingSphere
        Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[6]);
        Canvas.gl.scalef(testArea*0.5f, testArea*0.5f, testArea*0.5f);
        Canvas.gl.rotatef( -90.0f, 1.0f, 0.0f, 0.0f);
        Canvas.gl.callList(Canvas.plane);
        Canvas.gl.popMatrix(); // boundingSphere
    }
    //     velocityPointer
    if (velocityPointer) {
        float velocitySize = velocity.length() * 5.0f * testArea;
        Canvas.gl.pushMatrix(); // velocityPointer
        Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[10]);
        Canvas.gl.rotatef(90.0f, 1.0f, 0.0f, 0.0f);
        Canvas.gl.translatef(0.0f, velocitySize * 0.5f, 0.0f);
        Canvas.gl.scalef(velocitySize, velocitySize, velocitySize);
        Canvas.gl.callList(Canvas.plane);
        Canvas.gl.popMatrix(); // velocityPointer
    }
    //     IDNumber
    if (drawIDNumber) {
        float IDNumberSize = 1.0f;
        Canvas.gl.pushMatrix(); // drawIDNumber
        Canvas.gl.translatef(-IDNumberSize, 4.0f, 0.0f);
        Canvas.gl.pushMatrix(); // vorwahl
        Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[6]);
        Canvas.gl.rotatef(-90.0f, 1.0f, 0.0f, 0.0f);
        Canvas.gl.scalef(IDNumberSize, IDNumberSize, IDNumberSize);
        Canvas.gl.callList(Canvas.plane);
        Canvas.gl.scalef(0.2f,0.2f,0.2f);
        Canvas.gl.translatef(0.0f, -1.0f, 0.0f);
        Canvas.gl3DPrint(IDNumber.substring(0,4),1);
        Canvas.gl.popMatrix(); // vorwahl
        Canvas.gl.translatef(IDNumberSize*2.0f, 0.0f, 0.0f);
        Canvas.gl.pushMatrix(); // number
        Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[5]);
        Canvas.gl.rotatef(-90.0f, 1.0f, 0.0f, 0.0f);
        Canvas.gl.scalef(IDNumberSize, IDNumberSize, IDNumberSize);
        Canvas.gl.callList(Canvas.plane);
        Canvas.gl.scalef(0.2f,0.2f,0.2f);
        Canvas.gl.translatef(0.0f, -1.0f, 0.0f);
```

```
    Canvas.gl3DPrint(IDNumber.substring(4),1);
    Canvas.gl.popMatrix(); // number
    Canvas.gl.popMatrix(); // drawIDNumber
}
//     most recent added simthing?
if (IDNumber.equals(new String(SimthingMain.myIDNumber))) {
    //     texture and draw shadow
    Canvas.gl.pushMatrix(); // schattenSphere
    Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[1]);
    Canvas.gl.rotatef(90.0f, 1.0f, 0.0f, 0.0f);
    Canvas.gl.scalef(3f,3f,3f);
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.popMatrix(); // schattenSphere
}

//     draw FOV
if (Settings.DRAW_DEBUG) {
    Canvas.gl.disable(GL.TEXTURE_2D);
    Canvas.gl.color3f(0.0f, 0.0f, 0.0f);
    Canvas.gl.disable(GL.BLEND);
    Canvas.gl.enable(GL.DEPTH_TEST);
    Canvas.gl.disable(GL.LIGHTING);
    if (Settings.FOG)
        Canvas.gl.enable(GL.FOG);
    else
        Canvas.gl.disable(GL.FOG);
    //
    Canvas.gl.pushMatrix(); // FOV
    Canvas.gl.rotatef(FOV, 0.0f, 1.0f, 0.0f);
    Canvas.gl.begin(GL.LINES);
    Canvas.gl.vertex3f(0.0f, 0.0f, 0.0f);
    Canvas.gl.vertex3f( -2.0f, 0.0f, 0.0f);
    Canvas.gl.end();
    Canvas.gl.rotatef(FOV * -2.0f, 0.0f, 1.0f, 0.0f);
    Canvas.gl.begin(GL.LINES);
    Canvas.gl.vertex3f(0.0f, 0.0f, 0.0f);
    Canvas.gl.vertex3f( -2.0f, 0.0f, 0.0f);
    Canvas.gl.end();
    Canvas.gl.popMatrix(); // FOV
    }
    Canvas.gl.popMatrix(); // main
} // drawFooObject

// ---------------- //
// drawConnectLines
// ---------------- //
public void drawConnectLines() {

    // nearDumbObject
    if (nearDumbObject != null) {
        Canvas.gl.disable(GL.TEXTURE_2D);
        Canvas.gl.color3f(0.0f, 0.0f, 1.0f);
        Canvas.gl.disable(GL.BLEND);
        Canvas.gl.enable(GL.DEPTH_TEST);
        Canvas.gl.disable(GL.LIGHTING);
        if (Settings.FOG)
            Canvas.gl.enable(GL.FOG);
        else
            Canvas.gl.disable(GL.FOG);

        Canvas.gl.begin(GL.LINES);
        Canvas.gl.vertex3f(position.x, 0.0f, position.y);
        Canvas.gl.vertex3f(nearDumbObject.position.x, 0.0f,
                           nearDumbObject.position.y);
        Canvas.gl.end();
    }
    // nearMySimthing
    if (nearMySimthing != null) {
        Canvas.gl.disable(GL.TEXTURE_2D);
        Canvas.gl.color3f(0.0f, 0.0f, 1.0f);
        Canvas.gl.disable(GL.BLEND);
        Canvas.gl.enable(GL.DEPTH_TEST);
        Canvas.gl.disable(GL.LIGHTING);
        if (Settings.FOG)
            Canvas.gl.enable(GL.FOG);
        else
            Canvas.gl.disable(GL.FOG);
        //
        Canvas.gl.begin(GL.LINES);
        Canvas.gl.vertex3f(position.x, 0.0f, position.y);
        Canvas.gl.vertex3f(nearMySimthing.position.x, 0.0f,
                           nearMySimthing.position.y);
        Canvas.gl.end();
    }
    // nearASimthing
    if (nearASimthing != null) {
        Canvas.gl.disable(GL.TEXTURE_2D);
        Canvas.gl.color3f(0.0f, 0.0f, 1.0f);
        Canvas.gl.disable(GL.BLEND);
        Canvas.gl.enable(GL.DEPTH_TEST);
        Canvas.gl.disable(GL.LIGHTING);
        if (Settings.FOG)
            Canvas.gl.enable(GL.FOG);
        else
            Canvas.gl.disable(GL.FOG);
        //
        Canvas.gl.begin(GL.LINES);
        Canvas.gl.vertex3f(position.x, 0.0f, position.y);
        Canvas.gl.vertex3f(nearASimthing.position.x, 0.0f,
                           nearASimthing.position.y);
        Canvas.gl.end();
    }
}

// ---------------- //
// updateRelation
// ---------------- //
public void updateRelation() {
    //     setup an array to store relation to all the otherOnes
    relation = new Relation[pSimthingMain.aSimthingArray.size() - 1];
    int currentItem = 0;

    //     cycle through all ASimthings -> how is MySimthing done?
    if (Settings.RELATION_DEBUG)
        System.out.println();
    if (Settings.RELATION_DEBUG)
        System.out.println(IDNumber);
    for (int i = 0; i < pSimthingMain.aSimthingArray.size(); i++) {
        //     pointer to other ASimthing
        ASimthing otherOne = (ASimthing) pSimthingMain.aSimthingArray.get(i);
        //     not myself
        if (IDNumber != otherOne.IDNumber) {
            if (Settings.RELATION_DEBUG)
                System.out.println(" > " + otherOne.IDNumber);
//     create structure for relation
relation[currentItem] = new Relation(otherOne);

//     a/ we know the same people
for (int j = 0; j < numbers.length; j++) {
    //     look through all the numbers of otherOne
    for (int k = 0; k < otherOne.numbers.length; k++) {
        if (numbers[j].equals(otherOne.numbers[k])) {
            //     if it s the same write to current relation structure
            relation[currentItem].weKnowTheSame.add(new String("".valueOf(numbers[j])));
        }
    }
}

//     b/ i know you
for (int j = 0; j < numbers.length; j++) {
    //     find otherOne.IDNumber in numbers
    if (otherOne.IDNumber.equals(numbers[j])) {
        relation[currentItem].knowYou = true;
    }
}

//     c/ you know me
//     look through all the numbers of otherOne
for (int j = 0; j < otherOne.numbers.length; j++) {
    if (IDNumber.equals(otherOne.numbers[j])) {
        relation[currentItem].youKnowMe = true;
    }
}

//     d/ we know each other
if (relation[currentItem].youKnowMe == true
    && relation[currentItem].knowYou == true) {
    relation[currentItem].knowEachOther = true;
}

//     e/ amount of bekanntschaften
relation[currentItem].numberOfFriends = relation[currentItem].
    weKnowTheSame.size();
if (Settings.RELATION_DEBUG)
    System.out.println(" > numberOfFriends: " +
                       relation[currentItem].numberOfFriends +
                       " > " +
                       relation[currentItem].weKnowTheSame);
```

```
                if (Settings.RELATION_DEBUG)
                    System.out.println(“ > knowYou        : “ +
                                       relation[currentItem].knowYou);
                if (Settings.RELATION_DEBUG)
                    System.out.println(“ > youKnowMe      : “ +
                                       relation[currentItem].youKnowMe);
                if (Settings.RELATION_DEBUG)
                    System.out.println(“ > knowEachOther  : “ +
                                       relation[currentItem].knowEachOther);
                //     next item
                currentItem++;
            }
        }
    }

    // ---------------- //
    // Relation
    // structure for storing relation to ASimthing
    // ---------------- //
    public class Relation {

        public String IDNumber;
        public LinkedList weKnowTheSame = new LinkedList();
        public int numberOfFriends = 0;
        public boolean knowYou = false;
        public boolean youKnowMe = false;
        public boolean knowEachOther = false;
        public ASimthing pASimthing;

        public Relation(ASimthing tPASimthing) {
            pASimthing = tPASimthing;
            IDNumber = pASimthing.IDNumber;
        }
    }
}
```

MYSIMTHING *Diese Klasse stellt den Schwarm aus den Daten des Benutzers dar. Von ihr werden noch zwei Unterklassen gesteuert; ›Flock‹ und ›Moop‹.*

```
//
//
// simthing.simthingMain.mySimthing
//
//

package simthing;

import org.lwjgl.opengl.GL;

public class MySimthing {

    // ---------------------------------------------------------------- //
    // variables
    // ---------------------------------------------------------------- //
    public      Flock        pFlock;
    public      Vector2f     position;
    public      Vector2f     velocity;
    public      Vector2f     cursorPosition;
    private     String       gMode;
    private     Vector2f     atMyCursor           = new Vector2f(0, 0);
    private     int          localTime;
    public static String[]   numbers = null;
    // distance maps
    public      float        FOV                  = 30.0f;
    public      float        testArea             = 7.0f;
    public      float        minDistance          = 1.0f;
    // near dumbObject
    public      DumbObject   nearDumbObject       = null;
    public      float        distanceToDumbObject = 0;
    // near aSimthing
    public      ASimthing    nearASimthing        = null;
    public      float        distanceToASimthing  = 0;
    // ---------------- //
    // applySteering
    // ---------------- //
    private     float        max_force            = 0.1f;
    private     float        mass                 = 100.0f;
    private     float        max_speed            = 0.06f;

// ---------------------------------------------------------------- //
// constructors
// ---------------------------------------------------------------- //
public MySimthing(String [] tNumbers) {
    if (Settings.DEBUG)System.out.println(“* simthing.simthingMain.mySimthing”);
    // create objects
    numbers = tNumbers;
    pFlock = new Flock(tNumbers, this);
    cursorPosition = new Vector2f(0, 0);
    position = new Vector2f(10, 0);
    velocity = new Vector2f(0, 0);
}

// ---------------------------------------------------------------- //
// methods
// ---------------------------------------------------------------- //
public void cycle(String tGMode) {
    gMode = tGMode;
    if (gMode == “exploration”) {
        // ---------------- //
        // cycle()
        // ---------------- //
        pFlock.cycle();


        // inform flock about aSimthing
        if (nearASimthing != null) {
            pFlock.nearASimthing = nearASimthing;
        } else {
            pFlock.nearASimthing = null;
        }


        // ---------------- //
        // mySimthing velocity
        // ---------------- //

        // follow cursor
        // FOO / TEMP
        float mySpeed = 0.025f;
        velocity.sub(cursorPosition, position);
        velocity.scale(mySpeed);
        position.add(velocity);
```

```
// ---------------- //
// update cursor
// ---------------- //
// magnetic aSimthing
float cursorSpeed = 0.06f;
if (nearASimthing != null) {
    Vector2f cursorVelocity = new Vector2f();
    cursorVelocity.sub(nearASimthing.position, cursorPosition);
    cursorVelocity.scale(cursorSpeed);
    cursorPosition.add(cursorVelocity);
}
//     cursor
//     prepare drawing
Canvas.gl.enable(GL.TEXTURE_2D);
Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[2]);
Canvas.gl.enable(GL.BLEND);
Canvas.gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
Canvas.gl.color3f(1.0f, 1.0f, 1.0f);
Canvas.gl.disable(GL.DEPTH_TEST);
Canvas.gl.disable(GL.LIGHTING);
Canvas.gl.disable(GL.FOG); // ???
if (Settings.FOG)Canvas.gl.enable(GL.FOG);
else Canvas.gl.disable(GL.FOG);
//     position and draw
Canvas.gl.pushMatrix();   // cursor
Canvas.gl.translatef(cursorPosition.x, 0.0f, cursorPosition.y);
Canvas.gl.rotatef(-90.0f, 1.0f, 0.0f, 0.0f);
Canvas.gl.scalef(2.0f, 2.0f, 2.0f);
Canvas.gl.callList(Canvas.plane);
Canvas.gl.popMatrix();    // cursor

// model 0
//     prepare drawing
Canvas.gl.enable(GL.TEXTURE_2D);
Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[2]);
Canvas.gl.enable(GL.BLEND);
Canvas.gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
Canvas.gl.color3f(1.0f, 1.0f, 1.0f);
Canvas.gl.disable(GL.DEPTH_TEST);
Canvas.gl.disable(GL.LIGHTING);
if (Settings.FOG)Canvas.gl.enable(GL.FOG);
else Canvas.gl.disable(GL.FOG);
//
Canvas.gl.pushMatrix(); // mySimthing
Canvas.gl.translatef(position.x,
                     (float)Math.sin(Math.toRadians(localTime * 6)) / 4f + 1f,
                     position.y);
atMyCursor.sub(cursorPosition, position);
Canvas.gl.rotatef((float)Math.toDegrees(Math.atan2(atMyCursor.x, atMyCursor.y)),
                  0.0f, 1.0f, 0.0f);

if (Settings.DRAW_DEBUG)  {
    // draw FOV
    Canvas.gl.disable(GL.TEXTURE_2D);
    Canvas.gl.color3f(0.0f, 0.0f, 1.0f);
    Canvas.gl.disable(GL.BLEND);
    Canvas.gl.enable(GL.DEPTH_TEST);
    Canvas.gl.enable(GL.LIGHTING);
    Canvas.gl.enable(GL.FOG);
    //
    Canvas.gl.rotatef(FOV, 0.0f, 1.0f, 0.0f);
    Canvas.gl.begin(GL.LINES);
    Canvas.gl.vertex3f(0.0f, 0.0f, 0.0f);
    Canvas.gl.vertex3f(0.0f, 0.0f, 3.0f);
    Canvas.gl.end();
    Canvas.gl.rotatef(FOV * -2.0f, 0.0f, 1.0f, 0.0f);
    Canvas.gl.begin(GL.LINES);
    Canvas.gl.vertex3f(0.0f, 0.0f, 0.0f);
    Canvas.gl.vertex3f(0.0f, 0.0f, 3.0f);
    Canvas.gl.end();
}

Canvas.gl.popMatrix();   // mySimthing

if (Settings.DRAW_DEBUG)  {
    // nearDumbObject
    if (nearDumbObject != null) {
        Canvas.gl.disable(GL.TEXTURE_2D);
        Canvas.gl.color3f(0.0f, 0.0f, 0.0f);
        Canvas.gl.disable(GL.BLEND);
        Canvas.gl.enable(GL.DEPTH_TEST);
        Canvas.gl.enable(GL.LIGHTING);
        Canvas.gl.enable(GL.FOG);
        //
        Canvas.gl.begin(GL.LINES);
        Canvas.gl.vertex3f(position.x, 0.0f, position.y);
        Canvas.gl.vertex3f(nearDumbObject.position.x, 0.0f,
                           nearDumbObject.position.y);
        Canvas.gl.end();
    }
    // nearASimthing //// depends on nearDumbObject
    if (nearASimthing != null) {
        Canvas.gl.color3f(0.0f, 1.0f, 0.0f);
        Canvas.gl.begin(GL.LINES);
        Canvas.gl.vertex3f(position.x, 0.0f, position.y);
        Canvas.gl.vertex3f(nearASimthing.position.x, 0.0f,
                           nearASimthing.position.y);
        Canvas.gl.end();
        // ---------------- //
        // display aSimthing ID
        // ---------------- //
        Canvas.glOverlayPrint(20, 480-84, "" + nearASimthing.IDNumber, 1);
    }
  }
 }
 else {
     System.out.println("*** Error in mySimthingMain.cycle()");
 }
 localTime++;
}

// ---------------- //
// applySteering
// ---------------- //
private void applySteering(Vector2f new_steering) {
    // supply vector objects
    Vector2f steering_direction = new Vector2f();
    Vector2f steering_force = new Vector2f();
    Vector2f acceleration = new Vector2f();
    //     take the 'suggested' velocity modifikation as steering_direction
    steering_direction.set(new_steering);
    //     restrain it to the max_force
    steering_force.truncLength(max_force, steering_direction);
    //     apply the mass of the object
    acceleration.scale(1.0f / mass, steering_force);
    //     add it to the old velocity
    velocity.add(velocity, acceleration);
    //     restrain it to max_speed
    velocity.truncLength(max_speed);
    //     add velocity to current position
    position.add(velocity);
 }
}
```

FLOCK *Die Klasse ›Flock‹ steuert die Herde.*

```
//
//
// simthing.simthingMain.mySimthing.flock
//
//

package simthing;

public class Flock {

    /* ----------------------------------------------------------------- */
    // variables
    /* ----------------------------------------------------------------- */

    public MySimthing pMySimthing;
    private CDatabase gDatabase;
    public int numberOfMoops;
    public Moop[] pMoop;
    public float[][] moopMap;

    public String myMode = “idle”;
    public boolean doneSeeking = true;
    public int numberOfNeighbors;
    public String[] numbers;

    public ASimthing nearASimthing;

    /* ----------------------------------------------------------------- */
    // constructors
    /* ----------------------------------------------------------------- */

    public Flock(String [] tNumbers, MySimthing tPMySimthing) {
        if (Settings.DEBUG)System.out.println(“* simthing.simthingMain.mySimthing.flock”);
        // objects
        numbers = tNumbers;
        pMySimthing = tPMySimthing;
        // define initial variables
        numberOfMoops = numbers.length;
        numberOfNeighbors = (int) Math.min(5, numberOfMoops - 2);
        pMoop = new Moop[numberOfMoops];
        for (int i = 0; i < numberOfMoops; i++) { // create all Moops
            pMoop[i] = new Moop(i, this, numbers[i]);
        }
        moopMap = new float[numberOfMoops][numberOfMoops];
        updateMoopMap();
        System.out.println();

    }

    /* ----------------------------------------------------------------- */
    // methods
    /* ----------------------------------------------------------------- */
    /* ---------------- */
    // cycle()
    /* ---------------- */
    public void cycle() {
        // moops
        updateMoopMap();

        /* ---------------- */
        // draw object
        // they are actually drawn in moop
        /* ---------------- */
        for (int i = 0; i < numberOfMoops; i++) {
            pMoop[i].cycle();
        }
        // myMode
        if (myMode == “seek”) { // mySimthing.flock.seek
                                // check if done
            doneSeeking = false;
            for (int i = 0; i < numberOfMoops; i++) {
                if (pMoop[i].myMode == “flock”) {
                    doneSeeking = true;
                    myMode = “idle”;
                    if (Settings.DEBUG)System.out.println(“break @ “ + i);
                    //
                    break;
                }
            }
            //
        }
        else if (myMode == “idle”) { // mySimthing.flock.idle
                                     //
        }
        else {
            System.out.println(“*** Error in mySimthing.flock.cycle()”);
        }
    }

    /* ---------------- */
    // updateMoopMap()
    /* ---------------- */
    private void updateMoopMap() {
        // update Distance
        Vector2f d = new Vector2f();
        for (int y = 0; y < numberOfMoops; y++) {
            for (int x = y; x < numberOfMoops; x++) {
                if (x == y) {
                    moopMap[x][y] = -1.0f;
                }
                else {
                    d.sub(pMoop[y].position, pMoop[x].position);
                    moopMap[x][y] = (float) Math.sqrt(d.lengthSquared());
                    moopMap[y][x] = moopMap[x][y];
                }
            }
        }
    }
}
```

MOOP *Die kleinsten Elemente der Herde werden durch diese Klasse repräsentiert. Die Regeln die diese Menge aus Elementen letztendlich den Eindruck einer Herde vermitteln, werden hier definiert.*

```
//
//
// simthing.simthingMain.mySimthing.flock.moop
//
//

package simthing;

import org.lwjgl.opengl.GL;

public class Moop {

    /* ----------------------------------------------------------------- */
    // variables
    /* ----------------------------------------------------------------- */

    private             Flock       pFlock;
    private             int         ID;

    public              String      myMode      = “flock”;
    public              int[]       neighbors;

    public              int         localTime   = (int) (Math.random() * 360);

    public              Vector2f    position    = new Vector2f( (float) Math.random() * 10.0f,
                                                                (float) Math.random() * 10.0f);
    public              Vector2f    velocity    = new Vector2f(1.0f, 1.0f);
    private             Vector2f    v01, v02, v03, v04;
    private             Vector2f    steering_direction = new Vector2f();
    private             Vector2f    steering_force = new Vector2f();
    private             Vector2f    acceleration = new Vector2f();
    private             float       speed       = 0.05f;
    private             float       speed_offset = 0.005f;
    private             float       max_speed   = 0.12f;
    private             float       min_speed   = 0.03f;
    private             float       max_force   = 0.5f;
    private             float       mass        = 7.0f;
    private             float       orientation  = 0.0f;
    private             float       minDistance  = 1.0f;
    private             Vector2f    atMySimthing = new Vector2f();

    private float maximumDistance;

    private float hoppyPoppy = 60.0f;
    private float heightModifier = 1.0f;
    private float scaleModifier = 1.0f;

    private boolean isAFriend;

    public String IDNumber;

    /* ----------------------------------------------------------------- */
    // constructors
    /* ----------------------------------------------------------------- */

    public Moop(int tID, Flock tPFlock, String tIDNumber ) {
        pFlock = tPFlock;
        ID = tID;
        IDNumber = tIDNumber;
        neighbors = new int[pFlock.numberOfNeighbors];
        maximumDistance = (float)pFlock.numberOfMoops/15.0f + 5.0f;
    }

    /* ----------------------------------------------------------------- */
    // methods
    /* ----------------------------------------------------------------- */
    public void cycle() {

        localTime++;
        /* ---------------- */
        // cycle()
        /* ---------------- */

        // reset mods
        hoppyPoppy = 60;
        heightModifier = 1.0f;
        scaleModifier = 1.0f;
        mass         = 7.0f;
        max_speed    = 0.3f;
        isAFriend = true;

        // do something if aSimthin is around
        if (pFlock.nearASimthing != null) {
            isAFriend = false;
            for (int i = 0; i < pFlock.nearASimthing.numbers.length; i++) {
                if (IDNumber.equals(pFlock.nearASimthing.numbers[i])) {
                    mass         = 1.0f;
                    max_speed    = 1.0f;
                    scaleModifier = 1 + (float)Math.sin(Math.toRadians((localTime*4)%180))*2.0f;
                    isAFriend = true;
                    break;
                }
            }
            if (pFlock.nearASimthing.IDNumber.equals(IDNumber)) {
                hoppyPoppy = 80;
                heightModifier = 4.0f;
                mass         = 1.0f;
                max_speed    = 2.0f;
                scaleModifier = 2.0f;
                isAFriend = true;
            }
            // don t go too far
            Vector2f v = new Vector2f();
            v.sub(pFlock.pMySimthing.position, this.position);
            if (v.length()>maximumDistance){
                isAFriend = true;
            }
        }



        if (myMode == “flock”) { // mySimthing.flock.moop.flock

            // 00 determine neighbors
            updateNeighbors();

            // 01 avoid collision
            v01 = avoidCollision();
            v01.scale(0.1f);

            // 02 align velocity
            v02 = alignVelocity();
            v02.scale(0.002f);

            // 03 goto mySimthing
            v03 = gotoMySimthing();
            v03.scale(0.02f);

            // 04 gotoNeighbors
            v04 = gotoNeighbors();
            v04.scale(0.002f);

            steering_direction = new Vector2f();
            steering_direction.add(v01);
            steering_direction.add(v02);
            steering_direction.add(v03);
            steering_direction.add(v04);

            speed += (float) Math.random() * speed_offset - speed_offset / 2;
            if (speed > max_speed)
                speed = max_speed;
            if (speed < min_speed)
                speed = min_speed;

            steering_force.truncLength(max_force, steering_direction);
            acceleration.scale( (1.0f / mass), steering_force);
            velocity.add(acceleration);
            velocity.truncLength(speed);
            position.add(velocity);
            orientation = (float) Math.atan2( -velocity.x, velocity.y);

            drawMe();

        }
        else if (myMode == “seek”) { // mySimthing.flock.moop.seek
                                     //
        }
        else if (myMode == “eat”) { // mySimthing.flock.moop.eat

        }
        else {
            System.out.println(“*** Error in mySimthing.flock.moop.cycle()”);
        }
    }
```

```
/* ---------------- */
// clearNeighbors()
/* ---------------- */
public void clearNeighbors() {
    for (int i = 0; i < neighbors.length; i++) {
        neighbors[i] = -1;
    }
}

/* ---------------- */
// updateNeighbors()
/* ---------------- */
public void updateNeighbors() {

    clearNeighbors();

    int j = neighbors.length - 1;

    for (int i = 0; i < pFlock.moopMap[ID].length; i++) {
        if (pFlock.moopMap[ID][i] < minDistance && i != ID) {
            if (j >= 0) {
                neighbors[j--] = i;
            }
        }
    }

}

/* ---------------- */
// 01 avoidCollision()
/* ---------------- */
public Vector2f avoidCollision() {

    Vector2f v = new Vector2f();
    Vector2f w = new Vector2f();
    int j = 0;

    for (int i = 0; i < neighbors.length; i++) {
        if (neighbors[i] != -1 && pFlock.moopMap[ID][neighbors[i]] < minDistance) {
            w.sub(this.position, pFlock.pMoop[neighbors[i]].position);
            v.add(w);
            j++;
        }
    }

    if (j != 0) {
        v.scale(1.0f / j);
        v.normalize();
    }
    return v;
}

/* ---------------- */
// 02 alignVelocity()
/* ---------------- */
public Vector2f alignVelocity() {

    Vector2f v = new Vector2f();
    int j = 0;

    for (int i = 0; i < neighbors.length; i++) {
        if (neighbors[i] != -1) {
            v.add(pFlock.pMoop[neighbors[i]].velocity);
            j++;
        }
    }
    if (j != 0) {
        v.scale(1.0f / j);
        v.normalize();
    }
    return v;
}

/* ---------------- */
// 03 gotoMySimthing()
/* ---------------- */
public Vector2f gotoMySimthing() {

    Vector2f v = new Vector2f();
    v.sub(pFlock.pMySimthing.position, this.position);
    v.normalize();
    if (!isAFriend){
        v.negate();
        v.scale(0.2f);
    }
    return v;

}

/* ---------------- */
// 04 gotoNeighbors()
/* ---------------- */
public Vector2f gotoNeighbors() {

    Vector2f v = new Vector2f();
    int j = 0;

    for (int i = 0; i < neighbors.length; i++) {
        if (neighbors[i] != -1) {
            v.add(pFlock.pMoop[neighbors[i]].position);
            j++;
        }
    }
    if (j != 0) {
        v.scale(1.0f / j);
        v.normalize();
    }
    return v;
}

private void drawMe() {
    float height = heightModifier*0.5f - (((localTime * 3.2f) % hoppyPoppy - hoppyPoppy / 2.0f)
                                          * ((localTime * 3.2f) % hoppyPoppy - hoppyPoppy / 2.0f)) / (hoppyPoppy * hoppyPoppy)
/ 2)*heightModifier;

    /* ---------------- */
    // prepare object
    /* ---------------- */
    Canvas.gl.enable(GL.TEXTURE_2D);
    Canvas.gl.enable(GL.BLEND);
    Canvas.gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
    Canvas.gl.color3f(1.0f, 1.0f, 1.0f);
    Canvas.gl.disable(GL.DEPTH_TEST);
    Canvas.gl.disable(GL.LIGHTING);
    if (Settings.FOG)
        Canvas.gl.enable(GL.FOG);
    else
        Canvas.gl.disable(GL.FOG);
    /* ---------------- */
    // draw object
    /* ---------------- */
    float moopSize = 0.15f;
    // moop
    Canvas.gl.pushMatrix(); // moop
    Canvas.gl.translatef(position.x, height, position.y);
    Canvas.gl.rotatef( -90.0f, 1.0f, 0.0f, 0.0f);
    Canvas.gl.scalef(moopSize * scaleModifier, moopSize * scaleModifier,
                     moopSize * scaleModifier);
    Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[8]);
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.popMatrix(); // moop

    // moop_schatten
    Canvas.gl.pushMatrix(); // moop_schatten
    Canvas.gl.translatef(position.x, 0.0f, position.y);
    Canvas.gl.rotatef( -90.0f, 1.0f, 0.0f, 0.0f);
    Canvas.gl.scalef(moopSize * scaleModifier, moopSize * scaleModifier,
                     moopSize * scaleModifier);
    Canvas.gl.bindTexture(GL.TEXTURE_2D, Canvas.texture[17]);
    Canvas.gl.callList(Canvas.plane);
    Canvas.gl.popMatrix(); // moop_schatten

    Canvas.gl.end();
  }
}
```

USERINTERACTION *Die Klasse ›UserInteraction‹ wertet die verfügbaren Eingabegeräte aus und stellt die Ergebnisse anderen Klassen zur Verfügung.*

```
//
//
// simthing.simthingMain.UserInteraction
//
//

package simthing;

import org.lwjgl.input.*;

public class UserInteraction {

    public static boolean joypadAvailable   = true;
    public static boolean keyboardAvailable = true;
    // used to elimnate repeated input
    private static boolean OLD_BUTTON_01    = false;
    private static boolean OLD_BUTTON_02    = false;
    private static boolean OLD_LEFT         = false;
    private static boolean OLD_RIGHT        = false;


    public static void processInput() {
        if(keyboardAvailable)Keyboard.poll();
        if(joypadAvailable)Controller.poll();
        getInput();
    }

    public static void setup (){
        // controller
        System.out.print("* Creating controller > ");
        try {
            Controller.create();
        } catch (Exception e) {
            System.out.println("ERROR > "+e);
            joypadAvailable = false;
        }
        if(joypadAvailable)System.out.println("OK");

        // keyboard
        System.out.print("* Creating keyboard > ");
        try {
            Keyboard.create();
        } catch (Exception e) {
            System.out.println("ERROR > "+e);
        }
        if(keyboardAvailable)System.out.println("OK");
        if(!(keyboardAvailable||joypadAvailable))System.out.println("\n*** WARNING no inputdevices detected\n");
    }

    private static void getInput() {

        // update my specified keys
        SimthingMain.ROTATE_LEFT = false;
        SimthingMain.ROTATE_RIGHT = false;
        SimthingMain.FORWARD = false;
        SimthingMain.BACKWARDS = false;
        SimthingMain.LEFT = false;
        SimthingMain.RIGHT = false;
        // remove repeated input from buttons
        SimthingMain.BUTTON_01 = false;
        SimthingMain.BUTTON_02 = false;
        //SimthingMain.BUTTON_03 = false;
        //SimthingMain.BUTTON_04 = false;

        // ROCKFIRE joypad
        // _   _   5
        // _   4   2
        // 3   1   _
        // 0   _   _
        //
        // 6   8   7

        //joypad
        if (joypadAvailable) {
            if (Controller.y == -1000) { // up
                SimthingMain.FORWARD = true;
            }
            if (Controller.y == 1000) { // down
                SimthingMain.BACKWARDS = true;
            }
            if (Controller.x == -1000) { // left
                if (SimthingMain.gMode == "transition" || SimthingMain.gMode == "simulation" ) {
                    if (!OLD_LEFT) {
                        SimthingMain.LEFT = true;
                        OLD_LEFT = true;
                    }
                }
                else {
                    SimthingMain.LEFT = true;
                }
            }
            else {
                OLD_LEFT = false;
            }
            if (Controller.x == 1000) { // right
                if (SimthingMain.gMode == "transition" || SimthingMain.gMode == "simulation") {
                    if (!OLD_RIGHT) {
                        SimthingMain.RIGHT = true;
                        OLD_RIGHT = true;
                    }
                }
                else {
                    SimthingMain.RIGHT = true;
                }
            }
            else {
                OLD_RIGHT = false;
            }
            if (Controller.isButtonDown(3)) { // button good
                if (!OLD_BUTTON_01) {
                    SimthingMain.BUTTON_01 = true;
                    OLD_BUTTON_01 = true;
                }
            }
            else {
                OLD_BUTTON_01 = false;
            }
            if (Controller.isButtonDown(0)) { // button bad
                if (!OLD_BUTTON_02) {
                    SimthingMain.BUTTON_02 = true;
                    OLD_BUTTON_02 = true;
                }
            }
            else {
                OLD_BUTTON_02 = false;
            }
        } // if(joypadAvailable)
          //keyboard
        else if(keyboardAvailable){
            if (Keyboard.isKeyDown(Keyboard.KEY_Q)) { // button 6
                SimthingMain.ROTATE_LEFT = true;
            }
            if (Keyboard.isKeyDown(Keyboard.KEY_E)) { // button 7
                SimthingMain.ROTATE_RIGHT = true;
            }
            if (Keyboard.isKeyDown(Keyboard.KEY_W)) { // up
                SimthingMain.FORWARD = true;
            }
            if (Keyboard.isKeyDown(Keyboard.KEY_S)) { // down
                SimthingMain.BACKWARDS = true;
            }
            if (Keyboard.isKeyDown(Keyboard.KEY_A)) { // left
                if (SimthingMain.gMode == "transition" || SimthingMain.gMode == "simulation") {
                    if (!OLD_LEFT) {
                        SimthingMain.LEFT = true;
                        OLD_LEFT = true;
                    }
                }
                else {
                    SimthingMain.LEFT = true;
                }
            }
            else {
                OLD_LEFT = false;
            }
            if (Keyboard.isKeyDown(Keyboard.KEY_D)) { // right
                if (SimthingMain.gMode == "transition" || SimthingMain.gMode == "simulation") {
                    if (!OLD_RIGHT) {
                        SimthingMain.RIGHT = true;
                        OLD_RIGHT = true;
                    }
                }
                else {
                    SimthingMain.RIGHT = true;
                }
            }
            else {
                OLD_RIGHT = false;
```

```
            }
            if (Keyboard.isKeyDown(Keyboard.KEY_V)) { // button good
                if (!OLD_BUTTON_01) {
                    SimthingMain.BUTTON_01 = true;
                    OLD_BUTTON_01 = true;
                }
            }
            else {
                OLD_BUTTON_01 = false;
            }
            if (Keyboard.isKeyDown(Keyboard.KEY_G)) { // button bad
                if (!OLD_BUTTON_02) {
                    SimthingMain.BUTTON_02 = true;
                    OLD_BUTTON_02 = true;
                }
            }
            else {
                OLD_BUTTON_02 = false;
            }
        }// if(keyboardAvailable)
         // check for quit
        if (keyboardAvailable) {
            if (Keyboard.isKeyDown(Keyboard.KEY_ESCAPE)) {
                // exit
                SimthingMain.finished = true;
            }
            if (Keyboard.isKeyDown(Keyboard.KEY_SPACE)) {
                // screenshot
                if (SimthingMain.localTime%5==0)
                    SimthingMain.makeScreenshot();
            }
        } // check quit
    }// getInput
}// UserInteraction
```

DATAREADER *Diese Klasse liest die gespeicherten Daten von der Festplatte.*

```
//
//
// simthing.DataReader
//
//

package simthing;

import java.io.*;
import java.util.*;

public class DataReader {

    private static String directoryName = Settings.DATADIRECTORY;

    public static String[] load(String IDNumber) {
        Vector numberVector = new Vector();
        LineNumberReader f;
        String line = null;
        String number = null;
        StringTokenizer st = null;
        try {
            f = new LineNumberReader(new FileReader(directoryName + IDNumber));
            // read each line
            while ( (line = f.readLine()) != null) {
                // from NOW on there is only a number in each line
                number = line;
                // clean numbers 00xx ||†+xx
                if (number.charAt(0) == '0' && number.charAt(1) == '0') {
                    //System.out.print(IDNumber + " > " + number + " > ");
                    number = number.substring(4, number.length());
                    number = '0' + number;
                    //System.out.println(number);
                }
                else if (number.charAt(0) == '+') {
                    //System.out.print(IDNumber + " > " + number + " > ");
                    number = number.substring(3, number.length());
                    number = '0' + number;
                    //System.out.println(number);
                }
                else {
                    //System.out.println(IDNumber + " > " + number);
                }
                numberVector.addElement(number);
            }
            f.close();
        }
        catch (IOException e) {
            if (Settings.DEBUG) System.out.println("* ERROR reading data" + e);
        }
        String[] numberArray = new String[numberVector.size()];
        for (int i = 0; i < numberVector.size(); i++) {
            numberArray[i] = (String) numberVector.get(i);
        }
        return numberArray;
    }

    public static String[] getStored() {
        //     set directory name
        File dir = new File(directoryName);
        //     get files in directory
        File[] entries = dir.listFiles();
        //     temporary container
        Vector fileVector = new Vector();
        //     crawl the directory
        for (int i = 0; i < entries.length; ++i) {
            //     omit the ones with '.'
            //System.out.println("reading #" + i + ": " + entries[i]);
            if (entries[i].getName().charAt(0) != '.') {
                //     read the file
                fileVector.addElement(entries[i].getName());
            }
        }
        //     return list of stored data
        String[] fileArray = new String[fileVector.size()];
        for (int i = 0; i < fileVector.size(); i++) {
            fileArray[i] = (String) fileVector.get(i);
        }
        return fileArray;
    }
}
```

DATAWRITER *Diese Klasse schreibt Daten auf die Festplatte.*

```
//
//
// simthing.DataWriter
//
//

package simthing;

import java.io.*;

public class DataWriter {

    private static String directoryName = Settings.DATADIRECTORY;

    public static void write(String IDNumber, String[] userData) {
        BufferedWriter f;
        String s;
        // start writing data to disk
        try {
            f = new BufferedWriter(new FileWriter(directoryName+IDNumber));
            for (int i = 0; i < userData.length; ++i) {
                f.write(userData[i]);
                f.newLine();
            }
            f.close();
        }
        catch (IOException e) {
            if (Settings.DEBUG) System.out.println("* ERROR reading data" + e);
        }
    }
}
```

SIMCARDREADER *Diese Klasse realisiert die Kommunikation mit dem SIM Kartenleser.*

```
//
//
// simthing.SIMCardReader
/*
Lesen von Kurzrufnummern und Kurznachrichten aus einer SIM karte
 (c) http://www.wrankl.de/idx_jcrd.htm
 */
//
//

package simthing;

import org.lwjgl.*;
import org.lwjgl.opengl.*;
import org.lwjgl.input.*;

import java.util.*;
import opencard.core.service.*;
import opencard.core.terminal.*;
import opencard.opt.util.*;

import opencard.core.event.CTListener;
import opencard.core.event.CardTerminalEvent;
import opencard.core.event.EventGenerator;
import opencard.core.service.CardServiceException;
import opencard.core.service.SmartCard;
import opencard.core.terminal.CardTerminalException;
import opencard.core.terminal.CardTerminalRegistry;
import opencard.core.util.OpenCardPropertyLoadingException;

public class SIMCardReader
implements CTListener {

    // foo
    boolean cardReady = false;

    // errorhandling
    private static boolean error;
// waiting for card to be inserted
private static Object monitorInsert = "synchronization monitorInsert";
private static Object monitorRemove = "synchronization monitorRemove";

static final int THE_SAME = 0; // used for comparisons
static final int IFD_TIMEOUT = 0; // unit: seconds
static final int MAX_APDU_SIZE = 250; // unit: byte
static boolean DEBUG = false;

static final int CAPDU_P1 = 2; // pointer to P1 within a command APDU
static final int CAPDU_LE = 4; // pointer to Le within a command APDU

// command SELECT FILE with 2 byte FID, according to GSM 11.11
// CLA || INS || P1 || P2 || Lc || DATA 1 (= FID high) || DATA 2 (= FID low)
static final byte[] CMD_SELECT_BY_FID = {
    (byte) 0xA0, (byte) 0xA4,
    (byte) 0x00, (byte) 0x00, (byte) 0x02,
    (byte) 0x00, (byte) 0x00};

// command READ RECORD, according to GSM 11.11
// CLA || INS || P1 (record number) || P2 (mode) || Le (record length)
static final byte[] CMD_READ_RECORD = {
    (byte) 0xA0, (byte) 0xB2,
    (byte) 0x00, (byte) 0x04, (byte) 0x00};

// command GET RESPONSE, according to GSM 11.11
// CLA || INS || P1 || P2 || Le
static final byte[] CMD_GET_RESPONSE = {
    (byte) 0xA0, (byte) 0xC0,
    (byte) 0x00, (byte) 0x00, (byte) 0x00};

// command VERIFY CHV, according to GSM 11.11
// verify CHV 1 with CHV (= PIN) = "1234"
// CLA || INS || P1 || P2 || Lc || DATA 1 ... DATA 8 (= PIN)
// Format PIN: 4 byte ASCII left justified padded with 'FF'
static byte[] CMD_VERIFY_CHV = {
    (byte) 0xA0, (byte) 0x20,
    (byte) 0x00, (byte) 0x01, (byte) 0x08,
    (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
    (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF};

// FID of DF Telecom
static final byte[] FID_DF_TELECOM = {
    (byte) 0x7F, (byte) 0x10};
```

```
// FID of EF ADN
static final byte[] FID_EF_ADN = {
    (byte) 0x6F, (byte) 0x3A};

// FID of EF SMS
static final byte[] FID_EF_SMS = {
    (byte) 0x6F, (byte) 0x3C};

// Returncode '9Fxx': command propper executed, xx byte data available
static final byte RC_CMD_PROPPER_EXEC = (byte) 0x9F;

// pointer to the number of records (1 byte) within the response of a SELECT FILE
static final int R_SELECTFILE_NO_OF_RECORDS = 15 - 1;

// pointer to the file size (2 byte) within the response of a SELECT FILE
static final int R_SELECTFILE_FILE_SIZE = 3 - 1;

static private int n, i_fid;
static private int FileSize, RecordLength, NoOfRecords;
static private byte[] i;
private PassThruCardService ptcs;
private Properties sysProps;
static private ResponseAPDU response;
static private CommandAPDU command;

//
public String decodeADNRecord(byte[] RspAPDU, int LenRspAPDU) {
    int x, n, LenRecord, LenName, LenNumber;
    String b = new String();
    String c = new String();
    String personsName = new String("");
    String phoneNumber = new String();

    x = (int) (0x000000FF & RspAPDU[0]);
    if (x == 0xFF) {
        if (DEBUG)
            System.out.println("< void >");
        return personsName;
    }

    LenRecord = LenRspAPDU - 2; // length of the record = APDU length without SW1 || SW2
    LenName = LenRecord - 14;
    LenNumber = RspAPDU[LenName]; // read length of dialing number from the ADN record

    for (n = 0; n < LenName; n++) {
        x = (int) (0x000000FF & RspAPDU[n]);
        if (x == 0xFF)
            break; // end of dialing number reached
        personsName = personsName + (char) x;
    } // for

    for (n = LenName + 1; n < LenName + 1 + LenNumber; n++) {
        x = (int) (0x000000FF & RspAPDU[n]);
        if (x == 0xFF)
            break;
        c = Integer.toHexString(x).toUpperCase();
        if (c.length() == 1)
            c = "0" + c;
        if (n == (LenName + 1)) {
            // it is the 1st value of the dialing numbers
            if (c.compareTo("91") == THE_SAME)
                c = "+"; // it is an international dialing number
            if (c.compareTo("81") == THE_SAME)
                c = ""; // it is a standard (ISDN) dialing number
        } // if
        else {
            // if it is not the 1st value of the dialing number -> swap high and low digit
            b = c.substring(1, 2);
            if (! (c.substring(0, 1)).equals("F")) { // quick hack for the stoopif 'F' at the end of numbers
                b = b + c.substring(0, 1);
            }
            c = b;
        } // else
        phoneNumber = phoneNumber + c;
    } // for

    if (DEBUG)
        System.out.println(phoneNumber + "\t" + personsName);
    return phoneNumber;

} // decodeADNRecord
```

```
public String[] getData(String userPIN) { // hier PIN

    // errorhandling
    error = false;

    // vector to pass back the numbers
    Vector mobilNumbers = new Vector();

    // preapre PIN
    CMD_VERIFY_CHV[5] = (byte) userPIN.charAt(0);
    CMD_VERIFY_CHV[6] = (byte) userPIN.charAt(1);
    CMD_VERIFY_CHV[7] = (byte) userPIN.charAt(2);
    CMD_VERIFY_CHV[8] = (byte) userPIN.charAt(3);

    // get and set system properties for OCF, PC/SC and PassThruCardServce
    Properties sysProps = System.getProperties();
    sysProps.put("OpenCard.terminals",
                 "com.ibm.opencard.terminal.pcsc10.Pcsc10CardTerminalFactory");
    sysProps.put("OpenCard.services",
                 "opencard.opt.util.PassThruCardServiceFactory");

    try {
        // start smartcard operation
        if(SmartCard.isStarted()) SmartCard.shutdown(); // does this make sense?
        SmartCard.start();

        // add this object as a listener
        EventGenerator.getGenerator().addCTListener(this);

        // create smartcard object
        CardRequest cr = null;
        SmartCard sc = null;
        cr = new CardRequest(CardRequest.ANYCARD, null,
                             PassThruCardService.class);
        cr.setTimeout(IFD_TIMEOUT); // set timeout for IFD
        sc = SmartCard.waitForCard(cr); // wait  for ICC in the IFD

        // check if object was created otherwise ask for card
        if (sc == null) {
            // start UI listener
            cardReady = false;

            Canvas.startGL();
            // draw texture enter phone number
            if (Settings.GUI_CENTERED)
                Canvas.glOverlayAddUser(0, 0, 7, 512);
            else
                Canvas.glOverlayAddUser( 256-Settings.SCREEN[0]/2, 0, 7, 512);
            // vignette
            Canvas.glOverlayStretched(Settings.SCREEN[0] / 2,
                                      Settings.SCREEN[1] / 2, 19,
                                      Settings.SCREEN[0] + 2,
                                      Settings.SCREEN[1] + 2);
            Canvas.endGL();

            if (Settings.DEBUG)
                System.out.println("* insert card -->");

            // cycle UI listener and insert card request
            Sys.setProcessPriority(Sys.LOW_PRIORITY);
            while (!cardReady) {
                UserInteraction.processInput();
                if (SimthingMain.BUTTON_02) {
                    sc = null;
                    return new String[] {"error"};
                }
            }
            Sys.setProcessPriority(Sys.HIGH_PRIORITY);

            // creat card reader again
            cr = new CardRequest(CardRequest.ANYCARD, null,
                                 PassThruCardService.class);
            cr.setTimeout(IFD_TIMEOUT); // set timeout for IFD
            sc = SmartCard.waitForCard(cr); // wait  for ICC in the IFD

        } // if

        if (sc != null) { // no error occur, a smart card is in the terminal
            PassThruCardService ptcs = (PassThruCardService) sc.
            getCardService(PassThruCardService.class, true);
            command = new CommandAPDU(MAX_APDU_SIZE); // set APDU buffer size
                                                      // get ATR
            CardID cardID = sc.getCardID();
            i = cardID.getATR();
```

```
// ----- read and display all ADNs (abbreviated dialing numbers) -----
// select DF Telecom
command.setLength(0);
command.append(CMD_SELECT_BY_FID);
command.setByte(5, FID_DF_TELECOM[0]); // set FID high
command.setByte(6, FID_DF_TELECOM[1]); // set FID low
response = ptcs.sendCommandAPDU(command);
// VERIFY CHV ( = PIN)
command.setLength(0);
command.append(CMD_VERIFY_CHV);
response = ptcs.sendCommandAPDU(command);
// select EF ADN
command.setLength(0);
command.append(CMD_SELECT_BY_FID);
command.setByte(5, FID_EF_ADN[0]); // set FID high
command.setByte(6, FID_EF_ADN[1]); // set FID low
response = ptcs.sendCommandAPDU(command);
// send GET RESPONSE and fetch the data from the previous SELECT FILE command
command.setLength(0);
command.append(CMD_GET_RESPONSE);
command.setByte(4, response.getByte(1)); // set number of byte to fetch from card
response = ptcs.sendCommandAPDU(command);
command.setByte(4, response.getByte(1)); // set number of byte to fetch from card
FileSize = response.getByte(R_SELECTFILE_FILE_SIZE) * 256 +
    response.getByte(R_SELECTFILE_FILE_SIZE + 1);
RecordLength = response.getByte(R_SELECTFILE_NO_OF_RECORDS);
NoOfRecords = FileSize / RecordLength;

String ADNRecord = new String();
for (n = 1; n <= NoOfRecords; n++) { // loop, to read all records
                                    //     do card reading
    command.setLength(0);
    command.append(CMD_READ_RECORD);
    command.setByte(CAPDU_P1, n); // set record number
    command.setByte(CAPDU_LE, RecordLength); // set record length = byte to read
    response = ptcs.sendCommandAPDU(command);
    ADNRecord = decodeADNRecord(response.getBytes(),
                                response.getLength());
    if (ADNRecord.length() != 0) {
        mobilNumbers.add(ADNRecord); // System.out.println(ADNRecord);
        Canvas.startGL(); // end openGL
                          // draw texture enter phone number
        if (Settings.GUI_CENTERED) {
            Canvas.glOverlayAddUser(0, 0, 7, 512);
            Canvas.glOverlayPrint(ADNRecord.length() * -8, -16,
                                  ADNRecord, 1);
        }
        else {
            Canvas.glOverlayAddUser(256 -
                                    Settings.SCREEN[0] / 2, 0, 7, 512);
            Canvas.glOverlayPrint(256 - Settings.SCREEN[0] / 2 +
                                  ADNRecord.length() * -8, -16, ADNRecord, 1);
        }
        // vignette
        Canvas.glOverlayStretched(Settings.SCREEN[0] / 2,
                                  Settings.SCREEN[1] / 2, 19,
                                  Settings.SCREEN[0] + 2,
                                  Settings.SCREEN[1] + 2);
        Canvas.endGL(); // end openGL
    } // if
      //     check user interaction
    UserInteraction.processInput();
    if (SimthingMain.BUTTON_02) {
        if (Settings.DEBUG) System.out.println("* user cancelled card reading");
        error = true;
        break;
    } // if
} // for

synchronized (monitorRemove) {
    Canvas.startGL();
    if (Settings.GUI_CENTERED)
        Canvas.glOverlayAddUser(0, 0, 4, 512);
    else
        Canvas.glOverlayAddUser(256 - Settings.SCREEN[0] / 2, 0,
                                4, 512);
    // vignette
    Canvas.glOverlayStretched(Settings.SCREEN[0] / 2,
                              Settings.SCREEN[1] / 2, 19,
                              Settings.SCREEN[0] + 2,
                              Settings.SCREEN[1] + 2);
    Canvas.endGL();
                if (Settings.DEBUG)
                    System.out.println("* remove card -->");
                monitorRemove.wait();
            }

            SmartCard.shutdown();
        }
        else {
            if (Settings.DEBUG)
                System.out.println("* error \n* could not read from card");
            error = true;
        } // if
          //     clean up the sc object, gc is to slow
        sc = null;
    } // try
    catch (Exception except) {
        if (Settings.DEBUG)System.out.println("\n\nCaught exception '" + except.getClass() +
                                              "' - " + except.getMessage());
        error = true;
    } // catch
    String[] testArray = new String[mobilNumbers.size()];
    mobilNumbers.toArray(testArray);
    if (Settings.DEBUG)
        System.out.println("* card reading ok: " + !error + " -> " +
                           testArray.length + " numbers");
    if (error) {
        return new String[] {"error"};
    }
    else {
        return testArray;
    }
} // CardReader

/**
    * Gets invoked when a card is inserted.
    */
public void cardInserted(CardTerminalEvent ctEvent) {
    if (Settings.DEBUG)
        System.out.println("( card was inserted )");
        cardReady = true;
        //synchronized (monitorInsert) {
        //  monitorInsert.notify();
        //}
    } // cardInserted

    /**
        * Gets invoked when a card is removed. Is ignored in this case as we are
    * interested in getting events only when a card is being inserted.
        */
    public void cardRemoved(CardTerminalEvent ctEvent) {
        if (Settings.DEBUG)
            System.out.println("( card was removed )");
        synchronized (monitorRemove) {
            monitorRemove.notify();
        }
    } // cardRemoved
} // class
```

CANVAS *›Canvas‹ ist die OpenGL Klasse. Hier werden die wichtigsten Methoden zur Kommunikation mit der OpenGL Schnittstelle bereitgestellt. Eine Unterklasse von Canvas ist die ›Camera‹ Klasse.*

```
//
//
// simthing.simthingMain.Canvas
//
// some methods or lines were taken from
// NeHe tutorials
// and gametutorials.com
//
// * NeHe and GT, both rule make sure to check their tutorials
//
//

package simthing;

import org.lwjgl.*;
import org.lwjgl.opengl.*;
import org.lwjgl.input.*;

import java.util.ArrayList;
import java.lang.Math;
import java.nio.*;

import java.io.*;
import java.awt.geom.*;
import java.awt.Image;
import java.awt.Graphics2D;
import java.awt.image.*;

public class Canvas {

    /* ------------------------------------------------------------------ */
    // variables
    /* ------------------------------------------------------------------ */

    public static final GL gl = new GL(); // The Opengl Context
    public static final GLU glu = new GLU(gl); // The Opengl Utility Context

    public static CameraClass camera = new CameraClass(); // camera
    public static int[] texture; // texture
    public static int[] adduserTexture; // adduserTexture
                                        // primitives
    public static int cube; // cube
    public static int plane; // plane
    public static int fontList; // font
    public static int sphere; // sphere
    public static int border; // border
                              // light
    public float[] g_LightPosition = {
        1f, 5f, 0f, 0f};
    public float[] backgroundcolor = {
        1.0f, 1.0f, 1.0f, 0.0f};

    // timer
    private static long lastTime;
    private static int FPS;

    // byteBuffer
    public static ByteBuffer temp = ByteBuffer.allocateDirect(16);

    // frustumCulling
    public static CFrustum g_Frustum;

    public Canvas() {
        temp.order(ByteOrder.nativeOrder());
    }

    // camera follower stuff
    private static float oldCameraDistance;

    /* ------------------------------------------------------------------ */
    // methods
    /* ------------------------------------------------------------------ */

    /* -------------------------------- */
    // init
    /* -------------------------------- */

    public void init() {
        try {
            int mode = -1;
            DisplayMode[] modes = Display.getAvailableDisplayModes();
            for (int i = 0; i < modes.length; i++) {
                if (modes[i].width == Settings.SCREEN[0]
                    && modes[i].height == Settings.SCREEN[1]
                    && modes[i].bpp >= Settings.SCREEN[2]
                    && modes[i].freq == Settings.SCREEN[3]) {
                    if (Settings.DEBUG) {
                        System.out.println("Creating Display.... Width = "
                                           + Settings.SCREEN[0]
                                           + " - Height = " + Settings.SCREEN[1]
                                           + " - Bits = " + Settings.SCREEN[2]
                                           + " - Freq = " + Settings.SCREEN[3]);
                    }
                    mode = i;
                    break;
                }
            }
            Display.create(modes[mode], !Settings.DEBUG, "simthing");
        }
        catch (Exception e) {
            System.out.println("Error Creating Display");
            System.out.println(e);
            System.exit(1);
        }
        try {
            Keyboard.create();
        }
        catch (Exception e) {
            System.out.println("Error Creating Input");
            System.out.println(e);
        }
        try {
            gl.create();
        }
        catch (Exception e) {
            System.out.println("Error Creating OpenGL");
            System.out.println(e);
            System.exit(1);
        }

        // hog?
        if (Settings.RELATION_DEBUG)
            Sys.setProcessPriority(Sys.NORMAL_PRIORITY);
        else
            Sys.setProcessPriority(Sys.HIGH_PRIORITY);

        // timer
        Sys.setTime(0);
        if (Settings.DEBUG)
            System.out.println("Timer resolution: " + Sys.getTimerResolution());

        // enable environment
        enableEnvironment();

        // colors Smooth color Shading
        gl.shadeModel(GL.SMOOTH);
        //
        gl.depthFunc(GL.LEQUAL);
        // colors Clearing Of The Depth Buffer
        gl.clearDepth(1.0);
        // clearcolor
        gl.clearColor(backgroundcolor[0], backgroundcolor[1], backgroundcolor[2],
                      backgroundcolor[3]);
        // nice perspective
        gl.hint(GL.PERSPECTIVE_CORRECTION_HINT, GL.NICEST);

        // smooth edges / SLOW!
        // gl.enable(GL.POLYGON_SMOOTH);

        resetViewport(80.0f);

        // compile primitives
        cube = compileCube();
        plane = compilePlane();
        sphere = glu.newQuadric();
        glu.quadricDrawStyle(sphere, GLU.FILL);
        border = compileBorder();

        // setup camera
        camera.PositionCamera(0.0f, 15.5f, 6.0f,
                              0.0f, 1.5f, 0.0f,
                              0.0f, 1.0f, 0.0f);
        // frustumCulling
        g_Frustum = new CFrustum();

    }
```

```
/* --------------------------------- */
// load
// loading textures
/* --------------------------------- */
public static void resetViewport(float f){
    if (Settings.DEBUG)System.out.println("* resetingViewport");
    //Reset The Current Viewport And Perspective Transformation
    gl.viewport(0, 0, Display.getWidth(), Display.getHeight());
    gl.matrixMode(GL.PROJECTION);
    gl.loadIdentity();
    //              brennweite , ratio, ratio, depth culling
    glu.perspective(f, Display.getWidth() / Display.getHeight(), 1,
                    200.0f);
    gl.matrixMode(GL.MODELVIEW);
    gl.loadIdentity();
}

/* --------------------------------- */
// load
// loading textures
/* --------------------------------- */
public void prepareTextures() {
    if (Settings.DEBUG) System.out.println("* Loading textures");
    String textureNames[] = {
        "font.png",
        "center.png",
        "cursor.png",
        "harry.png",
        "harry_schatten.png",
        "kreis_breit.png",
        "kreis_schmal.png",
        "kugel_schatten.png",
        "moop.png",
        "rand.png",
        "richtung.png",
        "scanner01.png",
        "scanner02.png",
        "scanner03.png",
        "scanner04.png",
        "sleep.png",
        "vignette.png",
        "moop_schatten.png"
    };
    texture = loadTextures(textureNames, "mippedMap");

    String adduserDirectory = "adduser/";
    String adduserTextureNames[] = {
        adduserDirectory + "control_adduser.png",
        adduserDirectory + "control_exploration.png",
        adduserDirectory + "control_simulation.png",
        adduserDirectory + "doneexploring.png",
        adduserDirectory + "donereading.png",
        adduserDirectory + "phonenumber.png",
        adduserDirectory + "pinnumber.png",
        adduserDirectory + "readingdata.png",
        adduserDirectory + "errorreading.png",
        adduserDirectory + "startadduser.png",
        adduserDirectory + "userquit.png"
    };

    adduserTexture = loadTextures(adduserTextureNames, "mippedMap");

    // build font from texture
    buildFont();

}

public int[] loadTextures(String[] textureNames, String textureMode) {

    // Create A IntBuffer For Image Address In Memory
    IntBuffer buf = ByteBuffer.allocateDirect(4 * textureNames.length).
    order(ByteOrder.nativeOrder()).
    asIntBuffer();
    int bufPtr = Sys.getDirectBufferAddress(buf); // Get Image Address
    gl.genTextures(textureNames.length, bufPtr); // Create Texture In OpenGL

    int[] returnTexArray = new int[textureNames.length];

    for (int i = 0; i < textureNames.length; i++) {

        if (Settings.DEBUG)
            System.out.print("loading " + textureNames[i] + " -> ");

        Image image = (new javax.swing.ImageIcon(Settings.TEXTURE_PATH +
                                                 textureNames[i])).getImage();

        // Exctract The Image
        BufferedImage tex = new BufferedImage(image.getWidth(null),
                                              image.getHeight(null),
                                              BufferedImage.TYPE_3BYTE_BGR);
        Graphics2D g = (Graphics2D) tex.getGraphics();
        g.drawImage(image, null, null);
        g.dispose();

        // Flip Image
        AffineTransform tx = AffineTransform.getScaleInstance(1, -1);
        tx.translate(0, -image.getHeight(null));
        AffineTransformOp op = new AffineTransformOp(tx,
                                                     AffineTransformOp.
                                                     TYPE_NEAREST_NEIGHBOR);
        tex = op.filter(tex, null);

        // Put Image In Memory
        ByteBuffer scratch = ByteBuffer.allocateDirect(4 * tex.getWidth() *
                                                       tex.getHeight());
        int scratchAddress = Sys.getDirectBufferAddress(scratch);

        byte data[] = (byte[]) tex.getRaster().getDataElements(0, 0,
                                                               tex.getWidth(),
                                                               tex.getHeight(), null);
        scratch.clear();
        scratch.put(data);

        if (textureMode == "near") {
            // Create Nearest Filtered Texture
            gl.bindTexture(GL.TEXTURE_2D, buf.get(i));
            gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MIN_FILTER,
                             GL.NEAREST);
            gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MAG_FILTER,
                             GL.NEAREST);
            gl.texImage2D(GL.TEXTURE_2D, 0, GL.RGB, tex.getWidth(),
                          tex.getHeight(), 0,
                          GL.RGB, GL.UNSIGNED_BYTE, scratchAddress);
        }
        else if (textureMode == "linear") {
            // Create Linear Filtered Texture
            gl.bindTexture(GL.TEXTURE_2D, buf.get(i));
            gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MIN_FILTER,
                             GL.LINEAR);
            gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MAG_FILTER,
                             GL.LINEAR);
            gl.texImage2D(GL.TEXTURE_2D, 0, GL.RGB, tex.getWidth(),
                          tex.getHeight(), 0,
                          GL.RGB, GL.UNSIGNED_BYTE, scratchAddress);
        }
        else {
            // Create MipMapped Texture
            gl.bindTexture(GL.TEXTURE_2D, buf.get(i));
            gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MIN_FILTER,
                             GL.LINEAR);
            gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MAG_FILTER,
                             GL.LINEAR_MIPMAP_NEAREST);
            glu.build2DMipmaps(GL.TEXTURE_2D, 3, tex.getWidth(),
                               tex.getHeight(),
                               GL.RGB, GL.UNSIGNED_BYTE, scratchAddress);
        }

        returnTexArray[i] = buf.get(i);
        if (Settings.DEBUG)
            System.out.println("OK");

    }
    return returnTexArray; // Return Image Addresses In Memory
}

/* --------------------------------- */
// startGL
/* --------------------------------- */
public static void startGL() {
    gl.clear(GL.COLOR_BUFFER_BIT | GL.DEPTH_BUFFER_BIT); //Clear The Screen And The Depth Buffer
    gl.loadIdentity();
}
```

```
/* ---------------------------------- */
// endGL
/* ---------------------------------- */
public static void endGL() {
    gl.swapBuffers();
}

/* ---------------------------------- */
// prepare / step 1 of 2
/* ---------------------------------- */
public static void prepare() {

    long currentTime = System.currentTimeMillis();
    FPS++;
    if (Settings.DEBUG) {
        if ( (currentTime - lastTime) > 10000) {
            System.out.println("FPS >>> " + (FPS / 10));
            lastTime = currentTime;
            FPS = 0;
        }
    }

    //     check if viewport has changed
    if ( SimthingMain.gMode == "simulation" && SimthingMain.followCameraMinimumDistance!=oldCameraDistance) {
        resetViewport(50.0f + SimthingMain.followCameraMinimumDistance);
    }
    oldCameraDistance = SimthingMain.followCameraMinimumDistance;
    //     reset da machino
    startGL();

    // camera
    camera.Look();
    // frustumCulling
    g_Frustum.CalculateFrustum(Canvas.gl); //Reset The View

}

/* ---------------------------------- */
// update / step 2 of 2
/* ---------------------------------- */
public static void update() {
    drawDefaults();
    drawOverlay();
    endGL();
}

/* ---------------------------------- */
// primitves
// cube, plane
/* ---------------------------------- */

public void callListCube() {
    gl.callList(cube);
}

private int compileCube() {

    // Generate a unique id for our cube
    int list = gl.genLists(1);

    // We will compile the cube
    gl.newList(list, GL.COMPILE);

    gl.begin(GL.QUADS);
    // Front Face
    gl.normal3f(0.0f, 0.0f, 1.0f);
    gl.texCoord2f(0.0f, 0.0f);
    gl.vertex3f( -1.0f, -1.0f, 1.0f); // Bottom Left Of The Texture and Quad
    gl.texCoord2f(1.0f, 0.0f);
    gl.vertex3f(1.0f, -1.0f, 1.0f); // Bottom Right Of The Texture and Quad
    gl.texCoord2f(1.0f, 1.0f);
    gl.vertex3f(1.0f, 1.0f, 1.0f); // Top Right Of The Texture and Quad
    gl.texCoord2f(0.0f, 1.0f);
    gl.vertex3f( -1.0f, 1.0f, 1.0f); // Top Left Of The Texture and Quad
    // Back Face
    gl.normal3f(0.0f, 0.0f, -1.0f);
    gl.texCoord2f(1.0f, 0.0f);
    gl.vertex3f( -1.0f, -1.0f, -1.0f); // Bottom Right Of The Texture and Quad
    gl.texCoord2f(1.0f, 1.0f);
    gl.vertex3f( -1.0f, 1.0f, -1.0f); // Top Right Of The Texture and Quad
    gl.texCoord2f(0.0f, 1.0f);
    gl.vertex3f(1.0f, 1.0f, -1.0f); // Top Left Of The Texture and Quad
    gl.texCoord2f(0.0f, 0.0f);
    gl.vertex3f(1.0f, -1.0f, -1.0f); // Bottom Left Of The Texture and Quad
    // Top Face
    gl.normal3f(0.0f, 1.0f, 0.0f);
    gl.texCoord2f(0.0f, 1.0f);
    gl.vertex3f( -1.0f, 1.0f, -1.0f); // Top Left Of The Texture and Quad
    gl.texCoord2f(0.0f, 0.0f);
    gl.vertex3f( -1.0f, 1.0f, 1.0f); // Bottom Left Of The Texture and Quad
    gl.texCoord2f(1.0f, 0.0f);
    gl.vertex3f(1.0f, 1.0f, 1.0f); // Bottom Right Of The Texture and Quad
    gl.texCoord2f(1.0f, 1.0f);
    gl.vertex3f(1.0f, 1.0f, -1.0f); // Top Right Of The Texture and Quad
    // Bottom Face
    gl.normal3f(0.0f, -1.0f, 0.0f);
    gl.texCoord2f(1.0f, 1.0f);
    gl.vertex3f( -1.0f, -1.0f, -1.0f); // Top Right Of The Texture and Quad
    gl.texCoord2f(0.0f, 1.0f);
    gl.vertex3f(1.0f, -1.0f, -1.0f); // Top Left Of The Texture and Quad
    gl.texCoord2f(0.0f, 0.0f);
    gl.vertex3f(1.0f, -1.0f, 1.0f); // Bottom Left Of The Texture and Quad
    gl.texCoord2f(1.0f, 0.0f);
    gl.vertex3f( -1.0f, -1.0f, 1.0f); // Bottom Right Of The Texture and Quad
    // Right face
    gl.normal3f(1.0f, 0.0f, 0.0f);
    gl.texCoord2f(1.0f, 0.0f);
    gl.vertex3f(1.0f, -1.0f, -1.0f); // Bottom Right Of The Texture and Quad
    gl.texCoord2f(1.0f, 1.0f);
    gl.vertex3f(1.0f, 1.0f, -1.0f); // Top Right Of The Texture and Quad
    gl.texCoord2f(0.0f, 1.0f);
    gl.vertex3f(1.0f, 1.0f, 1.0f); // Top Left Of The Texture and Quad
    gl.texCoord2f(0.0f, 0.0f);
    gl.vertex3f(1.0f, -1.0f, 1.0f); // Bottom Left Of The Texture and Quad
    // Left Face
    gl.normal3f( -1.0f, 0.0f, 0.0f);
    gl.texCoord2f(0.0f, 0.0f);
    gl.vertex3f( -1.0f, -1.0f, -1.0f); // Bottom Left Of The Texture and Quad
    gl.texCoord2f(1.0f, 0.0f);
    gl.vertex3f( -1.0f, -1.0f, 1.0f); // Bottom Right Of The Texture and Quad
    gl.texCoord2f(1.0f, 1.0f);
    gl.vertex3f( -1.0f, 1.0f, 1.0f); // Top Right Of The Texture and Quad
    gl.texCoord2f(0.0f, 1.0f);
    gl.vertex3f( -1.0f, 1.0f, -1.0f); // Top Left Of The Texture and Quad
    gl.end();

    gl.endList();

    return list;
}

private int compilePlane() {
    // Generate a unique id for our plane
    int list = gl.genLists(1);

    // We will compile the plane
    gl.newList(list, GL.COMPILE);

    gl.begin(GL.QUADS);
    gl.normal3f(0.0f, 0.0f, 1.0f);
    gl.texCoord2f(0.0f, 0.0f);
    gl.vertex3f( -1.0f, -1.0f, 0.0f); // Bottom Left Of The Texture and Quad
    gl.texCoord2f(1.0f, 0.0f);
    gl.vertex3f(1.0f, -1.0f, 0.0f); // Bottom Right Of The Texture and Quad
    gl.texCoord2f(1.0f, 1.0f);
    gl.vertex3f(1.0f, 1.0f, 0.0f); // Top Right Of The Texture and Quad
    gl.texCoord2f(0.0f, 1.0f);
    gl.vertex3f( -1.0f, 1.0f, 0.0f); // Top Left Of The Texture and Quad
    gl.end();

    gl.endList();

    return list;
}

private int compileBorder() {
    // Generate a unique id for our plane
    int list = gl.genLists(1);

    // We will compile the plane
    gl.newList(list, GL.COMPILE);
    for (int j = 0; j < 4; j++) {
        gl.pushMatrix();
        gl.translatef(Settings.TERRAIN_SIZE * 2.0f - 1, 2.0f, Settings.TERRAIN_SIZE * 2.0f);
        for (int i = - (int) Settings.TERRAIN_SIZE;
             i < (int) Settings.TERRAIN_SIZE - 1; i++) {
            gl.translatef(0.0f, 0.0f, -2.0f);
            gl.pushMatrix();
```

```
            gl.rotatef(90.0f, 0.0f, 1.0f, 0.0f);
            gl.callList(plane);
            gl.popMatrix();
        }
        gl.popMatrix();
        gl.rotatef(90.0f, 0.0f, 1.0f, 0.0f);
    }
    gl.endList();
    return list;
}


/* -------------------------------- */
// buildFont()
/* -------------------------------- */

private final static void buildFont() {
    float cx; // Holds Our X Character Coord
    float cy; // Holds Our Y Character Coord

    fontList = gl.genLists(256); // Creating 256 Display Lists
    gl.bindTexture(GL.TEXTURE_2D, texture[0]); // Select Our Font Texture

    for (int i = 0; i < 256; i++) { // Loop Through All 256 Lists
        cx = (float) (i % 16) / 16.0f; // X Position Of Current Character
        cy = (float) (i / 16) / 16.0f; // Y Position Of Current Character

        int fontSize = 32;
        int fontSpacing = 16;

        gl.newList(fontList + i, GL.COMPILE); // Start Building A List
        gl.begin(GL.QUADS); // Use A Quad For Each Character
        gl.texCoord2f(cx, 1 - cy - 0.0625f); // Texture Coord (Bottom Left)
        gl.vertex2i(0, 0); // Vertex Coord (Bottom Left)
        gl.texCoord2f(cx + 0.0625f, 1 - cy - 0.0625f); // Texture Coord (Bottom Right)
        gl.vertex2i(fontSize, 0); // Vertex Coord (Bottom Right)
        gl.texCoord2f(cx + 0.0625f, 1 - cy); // Texture Coord (Top Right)
        gl.vertex2i(fontSize, fontSize); // Vertex Coord (Top Right)
        gl.texCoord2f(cx, 1 - cy); // Texture Coord (Top Left)
        gl.vertex2i(0, fontSize); // Vertex Coord (Top Left)
        gl.end(); // Done Building Our Quad (Character)
        gl.translated(fontSpacing, 0, 0); // Move To The Right Of The Character
        gl.endList(); // Done Building The Display List
    } // Loop Until All 256 Are Built
}


/* -------------------------------- */
// killFont()
/* -------------------------------- */
public final static void killFont() {
    gl.deleteLists(fontList, 256); // Delete All 256 Display Lists
}


/* -------------------------------- */
// glOverlayPrint(int x, int y, String string, int set)
/* -------------------------------- */
public final static void glOverlayAddUser(int x, int y, int textureID, int size) {
    glOverlay( Settings.SCREEN[0] / 2 + x,
               Settings.SCREEN[1] / 2 + y,
               adduserTexture[textureID], size);
}


/* -------------------------------- */
// glOverlayPrint(int x, int y, String string, int set)
/* -------------------------------- */
public final static void glOverlay(int x, int y, int myTexture, int size) {
    /* ---------------- */
    // prepare object
    /* ---------------- */
    gl.enable(GL.TEXTURE_2D);
    gl.bindTexture(GL.TEXTURE_2D, myTexture);
    gl.color3f(1.0f, 1.0f, 1.0f);
    gl.enable(GL.BLEND);
    gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
    gl.disable(GL.DEPTH_TEST);
    gl.disable(GL.LIGHTING);
    gl.disable(GL.FOG);

    gl.matrixMode(GL.PROJECTION); // Select The Projection Matrix
    gl.pushMatrix(); // Store The Projection Matrix
    gl.loadIdentity(); // Reset The Projection Matrix
    gl.ortho(0, Display.getWidth(), 0, Display.getHeight(), -1, 1);
    gl.matrixMode(GL.MODELVIEW); // Select The Modelview Matrix
    gl.pushMatrix(); // Store The Modelview Matrix
    gl.loadIdentity(); // Reset The Modelview Matrix
    gl.translated(x, y, 0); // Position The Text (0,0 - Bottom Left)
    gl.scalef(size/2, size/2, size/2);
    gl.callList(plane);
    gl.matrixMode(GL.PROJECTION); // Select The Projection Matrix
    gl.popMatrix(); // Restore The Old Projection Matrix
    gl.matrixMode(GL.MODELVIEW); // Select The Modelview Matrix
    gl.popMatrix(); // Restore The Old Projection Matrix

}


/* -------------------------------- */
// glOverlayStretched(int x, int y, String string, int set)
/* -------------------------------- */
public final static void glOverlayStretched(int x, int y, int myTexture, int xSize, int ySize) {
    /* ---------------- */
    // prepare object
    /* ---------------- */
    gl.enable(GL.TEXTURE_2D);
    gl.bindTexture(GL.TEXTURE_2D, myTexture);
    gl.color3f(1.0f, 1.0f, 1.0f);
    gl.enable(GL.BLEND);
    gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
    gl.disable(GL.DEPTH_TEST);
    gl.disable(GL.LIGHTING);
    gl.disable(GL.FOG);

    gl.matrixMode(GL.PROJECTION); // Select The Projection Matrix
    gl.pushMatrix(); // Store The Projection Matrix
    gl.loadIdentity(); // Reset The Projection Matrix
    gl.ortho(0, Display.getWidth(), 0, Display.getHeight(), -1, 1);
    gl.matrixMode(GL.MODELVIEW); // Select The Modelview Matrix
    gl.pushMatrix(); // Store The Modelview Matrix
    gl.loadIdentity(); // Reset The Modelview Matrix
    gl.translated(x, y, 0); // Position The Text (0,0 - Bottom Left)
    gl.scalef(xSize/2, ySize/2, 1.0f);
    gl.callList(plane);
    gl.matrixMode(GL.PROJECTION); // Select The Projection Matrix
    gl.popMatrix(); // Restore The Old Projection Matrix
    gl.matrixMode(GL.MODELVIEW); // Select The Modelview Matrix
    gl.popMatrix(); // Restore The Old Projection Matrix
}


/* -------------------------------- */
// glStaticPrint(int x, int y, String string, int set)
/* -------------------------------- */
public final static void glStaticPrint(int x, int y, String string, int set) {
    startGL();
    glOverlayPrint(x, y, string, set);
    camera.Look();
    drawDefaults();
    endGL();
}


/* -------------------------------- */
// glOverlayPrint(int x, int y, String string, int set)
/* -------------------------------- */
public final static void glOverlayPrint(int x, int y, String string,
                                        int set) {
    x += Settings.SCREEN[0] / 2;
    y += Settings.SCREEN[1] / 2;
    if (set > 1)
        set = 1;
    /* ---------------- */
    // prepare object
    /* ---------------- */
    gl.enable(GL.TEXTURE_2D);
    gl.bindTexture(GL.TEXTURE_2D, texture[0]);
    gl.color3f(1.0f, 1.0f, 1.0f);
    gl.enable(GL.BLEND);
    //gl.blendFunc(GL.ZERO, GL.ONE_MINUS_SRC_COLOR);
    gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
    gl.disable(GL.DEPTH_TEST);
    gl.disable(GL.LIGHTING);
    gl.disable(GL.FOG);

    gl.matrixMode(GL.PROJECTION); // Select The Projection Matrix
    gl.pushMatrix(); // Store The Projection Matrix
```

```
    gl.loadIdentity(); // Reset The Projection Matrix
    gl.ortho(0, Display.getWidth(), 0, Display.getHeight(), -1, 1); // Set Up An Ortho Screen
    gl.matrixMode(GL.MODELVIEW); // Select The Modelview Matrix
    gl.pushMatrix(); // Store The Modelview Matrix
    gl.loadIdentity(); // Reset The Modelview Matrix
    gl.translated(x, y, 0); // Position The Text (0,0 - Bottom Left)
    gl.listBase(fontList - 32 + (128 * set)); // Choose The Font Set (0 or 1)

    ByteBuffer scratch = ByteBuffer.allocateDirect(string.getBytes().length);
    scratch.put(string.getBytes());
    gl.callLists(string.length(), GL.BYTE,
                 Sys.getDirectBufferAddress(scratch)); // Write The Text To The Screen

    gl.matrixMode(GL.PROJECTION); // Select The Projection Matrix
    gl.popMatrix(); // Restore The Old Projection Matrix
    gl.matrixMode(GL.MODELVIEW); // Select The Modelview Matrix
    gl.popMatrix(); // Restore The Old Projection Matrix

}

/* --------------------------------- */
// gl3DPrint(int x, int y, String string, int set)
/* --------------------------------- */

public final static void gl3DPrint(String string, int set) {
    if (set > 1)
        set = 1;

    /* ---------------- */
    // prepare object
    /* ---------------- */
    gl.enable(GL.TEXTURE_2D);
    gl.bindTexture(GL.TEXTURE_2D, texture[0]);
    gl.color3f(1.0f, 1.0f, 1.0f);
    gl.enable(GL.BLEND);
    gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
    gl.disable(GL.DEPTH_TEST);
    gl.disable(GL.LIGHTING);
    gl.disable(GL.FOG);
    gl.pushMatrix();
    gl.scalef(0.05f, 0.05f, 0.05f);
    gl.translatef( (float) string.length() * -10, 0f, 0f);
    gl.listBase(fontList - 32 + (128 * set)); // Choose The Font Set (0 or 1)
    ByteBuffer scratch = ByteBuffer.allocateDirect(string.getBytes().length);
    scratch.put(string.getBytes());
    gl.callLists(string.length(), GL.BYTE,
                 Sys.getDirectBufferAddress(scratch)); // Write The Text To The Screen
    gl.popMatrix();

}

/* --------------------------------- */
// enableFog()
/* --------------------------------- */
private void enableEnvironment() {
    //enables GL.FOG
    gl.enable(GL.FOG);
    //Fog Mode
    gl.fogi(GL.FOG_MODE, GL.EXP2);
    //Set Fog color
    gl.fogfv(GL.FOG_COLOR,
             Sys.getDirectBufferAddress(temp.asFloatBuffer().
                                        put(backgroundcolor)));
    //How Dense Will The Fog Be
    gl.fogf(GL.FOG_DENSITY, 0.04f);
    //Fog Hint Value
    gl.hint(GL.FOG_HINT, GL.DONT_CARE);
    //Fog Start Depth
    gl.fogf(GL.FOG_START, 40.0f);
    //Fog End Depth
    gl.fogf(GL.FOG_END, 50.0f);

}

/* --------------------------------- */
// drawBackdrop()
/* --------------------------------- */
private void drawBackdrop() {
}
```

```
    /* --------------------------------- */
    // drawOverlay()
    /* --------------------------------- */
    private static void drawOverlay() {
    }


    /* --------------------------------- */
    // drawDefaults()
    /* --------------------------------- */
    private static void drawDefaults() {
        //     vignette
        glOverlayStretched(Settings.SCREEN[0]/2, Settings.SCREEN[1]/2, 19, Settings.SCREEN[0]+2, Settings.SCREEN[1]+2);
        //     draw border
        gl.enable(GL.TEXTURE_2D);
        gl.bindTexture(GL.TEXTURE_2D, texture[9]);
        gl.enable(GL.BLEND);
        gl.blendFunc(GL.ZERO, GL.SRC_COLOR);
        gl.color3f(1.0f, 1.0f, 1.0f);
        gl.disable(GL.DEPTH_TEST);
        gl.disable(GL.LIGHTING);
        if (Settings.FOG)
            gl.enable(GL.FOG);
        else
            gl.disable(GL.FOG);
        gl.pushMatrix();
        gl.translatef(0.0f,5.0f,0.0f);
        gl.callList(border);
        gl.popMatrix();
        //     draw center
        gl.bindTexture(GL.TEXTURE_2D, texture[1]);
        gl.pushMatrix();
        gl.rotatef(90.0f, 1.0f, 0.0f, 0.0f);
        gl.scalef(4f,4f,4f);
        gl.callList(plane);
        gl.popMatrix();
    }
}
```

CAMERACLASS *Diese Klasse stammt aus einem Tutorial von ›gametutorials‹. Sie stellt eine simple Kamera zur Verfügung.*

```
//
//
// simthing.CameraClass
//
// from gametutorials.com
//
// *GT rule make sure to check their tutorials
//
//

package simthing;

import org.lwjgl.opengl.*;

public class CameraClass {
  /* ------------------------------------------------------------------ */
  // variables
  /* ------------------------------------------------------------------ */

  CVector3 vZero = new CVector3(0.0f, 0.0f, 0.0f); // Init a vector to 0 0 0 for our position
  CVector3 vView = new CVector3(0.0f, 1.0f, 0.5f); // Init a starting view vector (looking up and out the screen)
  CVector3 vUp = new CVector3(0.0f, 0.0f, 1.0f); // Init a standard up vector (Rarely ever changes)
  CVector3 m_vPosition = vZero; // Init the position to zero
  CVector3 m_vView = vView; // Init the view to a std starting view
  CVector3 m_vUpVector = vUp; // Init the UpVector
  CVector3 m_vStrafe = new CVector3();

  /* ------------------------------------------------------------------ */
  // constructors
  /* ------------------------------------------------------------------ */
  public CameraClass() {
    if (Settings.DEBUG) System.out.println(“* simthing.simthingMain.canvas.camera”);
  }

  /* ------------------------------------------------------------------ */
  // methods
  /* ------------------------------------------------------------------ */
  /* --------------------------------- */
  // PositionCamera ( position, view, up )
  // This function sets the camera’s position and view and up vector.
  /* --------------------------------- */
  public void PositionCamera(float positionX, float positionY,
·····                      float positionZ,
·····                      float viewX, float viewY, float viewZ,
·····                      float upVectorX, float upVectorY,
·····                      float upVectorZ) {
    CVector3 vPosition = new CVector3(positionX, positionY, positionZ);
    CVector3 vView = new CVector3(viewX, viewY, viewZ);
    CVector3 vUpVector = new CVector3(upVectorX, upVectorY, upVectorZ);
    m_vPosition = vPosition; // Assign the position
    m_vView = vView; // Assign the view
    m_vUpVector = vUpVector; // Assign the up vector
  }

  /* --------------------------------- */
  // PointCamera
  /* --------------------------------- */
  public void PointCamera(float viewX, float viewY, float viewZ) {
    CVector3 vView = new CVector3(viewX, viewY, viewZ);
    m_vView = vView;
  }

  /* --------------------------------- */
  // MoveCamera ( speed )
  // This will move the camera forward or backward depending on the speed
  /* --------------------------------- */
  public void MoveCamera(float speed) {
    CVector3 vVector = new CVector3(); // Init a vector for our view
    // Get our view vector (The direciton we are facing)
    vVector.x = m_vView.x - m_vPosition.x; // This gets the direction of the X
    vVector.y = m_vView.y - m_vPosition.y; // This gets the direction of the Y
    vVector.z = m_vView.z - m_vPosition.z; // This gets the direction of the Z
    vVector = normalize(vVector);
    m_vPosition.x += vVector.x * speed; // Add our acceleration to our position’s X
    m_vPosition.z += vVector.z * speed; // Add our acceleration to our position’s Z
    m_vView.x += vVector.x * speed; // Add our acceleration to our view’s X
    m_vView.z += vVector.z * speed; // Add our acceleration to our view’s Z
  }

  /* --------------------------------- */
  // RotateView ( angle ?radians or degree? , position )
  // This rotates the view around the position using an axis-angle rotation
  /* --------------------------------- */
  public void RotateView(float angle, float x, float y, float z) {
    CVector3 vNewView = new CVector3();
    CVector3 vView = new CVector3();
    // Get our view vector (The direciton we are facing)
    vView.x = m_vView.x - m_vPosition.x; // This gets the direction of the X
    vView.y = m_vView.y - m_vPosition.y; // This gets the direction of the Y
    vView.z = m_vView.z - m_vPosition.z; // This gets the direction of the Z
    // Calculate the sine and cosine of the angle once
    float cosTheta = (float) Math.cos(angle);
    float sinTheta = (float) Math.sin(angle);
    // Find the new x position for the new rotated point
    vNewView.x = (cosTheta + (1 - cosTheta) * x * x) * vView.x;
    vNewView.x += ( (1 - cosTheta) * x * y - z * sinTheta) * vView.y;
    vNewView.x += ( (1 - cosTheta) * x * z + y * sinTheta) * vView.z;
    // Find the new y position for the new rotated point
    vNewView.y = ( (1 - cosTheta) * x * y + z * sinTheta) * vView.x;
    vNewView.y += (cosTheta + (1 - cosTheta) * y * y) * vView.y;
    vNewView.y += ( (1 - cosTheta) * y * z - x * sinTheta) * vView.z;
    // Find the new z position for the new rotated point
    vNewView.z = ( (1 - cosTheta) * x * z - y * sinTheta) * vView.x;
    vNewView.z += ( (1 - cosTheta) * y * z + x * sinTheta) * vView.y;
    vNewView.z += (cosTheta + (1 - cosTheta) * z * z) * vView.z;
    // Now we just add the newly rotated vector to our position to set
    // our new rotated view of our camera.
    m_vView.x = m_vPosition.x + vNewView.x;
    m_vView.y = m_vPosition.y + vNewView.y;
    m_vView.z = m_vPosition.z + vNewView.z;
  }

  /* --------------------------------- */
  // StrafeCamera ( speed )
  // This strafes the camera left and right depending on the speed
  /* --------------------------------- */
  public void StrafeCamera(float speed) {
    // Get the normalized strafe vector
    CVector3 vCross = Cross(Subtract(m_vView, m_vPosition), m_vUpVector);
    m_vStrafe = normalize(vCross);
    // Add the strafe vector to our position
    m_vPosition.x += m_vStrafe.x * speed;
    m_vPosition.z += m_vStrafe.z * speed;
    // Add the strafe vector to our view
    m_vView.x += m_vStrafe.x * speed;
    m_vView.z += m_vStrafe.z * speed;
  }

  /* --------------------------------- */
  // Look
  // This updates the camera
  /* --------------------------------- */
  public void Look() {
    // camera props to openGL
    /*
     pCanvas.setCamera(m_vPosition.x, m_vPosition.y, m_vPosition.z,
·····         m_vView.x,   m_vView.y,     m_vView.z,
·····         m_vUpVector.x, m_vUpVector.y, m_vUpVector.z);
     */
    Canvas.glu.lookAt(m_vPosition.x, m_vPosition.y, m_vPosition.z,
       m_vView.x, m_vView.y, m_vView.z,
       m_vUpVector.x, m_vUpVector.y, m_vUpVector.z);
  }

  /* ------------------------------------------------------------------ */
  // 3D vector class
  // + some methods for working with vector 3D
  // in a future version they will be own class like the 2D stuff
  /* ------------------------------------------------------------------ */
  class CVector3 {
    float x;
    float y;
    float z;

    CVector3() {
      this.x = 0.0f;
      this.y = 0.0f;
      this.z = 0.0f;
    }

    CVector3(float x, float y, float z) {
      this.x = x;
      this.y = y;
```

```
      this.z = z;
    }
  }

  /* ---------------------------------- */
  // Cross (vector, vector)
  // returns a perpendicular vector
  /* ---------------------------------- */
  private CVector3 Cross(CVector3 vVector1, CVector3 vVector2) {
    CVector3 vnormal = new CVector3();
    // The X value for the vector is:  (V1.y * V2.z) - (V1.z * V2.y)
    vnormal.x = ( (vVector1.y * vVector2.z) - (vVector1.z * vVector2.y));
    // The Y value for the vector is:  (V1.z * V2.x) - (V1.x * V2.z)
    vnormal.y = ( (vVector1.z * vVector2.x) - (vVector1.x * vVector2.z));
    // The Z value for the vector is:  (V1.x * V2.y) - (V1.y * V2.x)
    vnormal.z = ( (vVector1.x * vVector2.y) - (vVector1.y * vVector2.x));
    return vnormal;
  }

  /* ---------------------------------- */
  // Magnitude ( vector )
  // returns the magnitude of a vector
  /* ---------------------------------- */
  private float Magnitude(CVector3 vnormal) {
    return (float) Math.sqrt( (vnormal.x * vnormal.x) +
.........                   (vnormal.y * vnormal.y) +
.........                   (vnormal.z * vnormal.z));
  }

  /* ---------------------------------- */
  // normalize ( vector )
  // returns a normalized vector
  /* ---------------------------------- */
  private CVector3 normalize(CVector3 vVector) {
    float magnitude = Magnitude(vVector);
    vVector.x /= magnitude;
    vVector.y /= magnitude;
    vVector.z /= magnitude;
    return vVector;
  }

  private CVector3 Add(CVector3 v1, CVector3 v2) {
    return new CVector3(v1.x + v2.x, v1.y + v2.y, v1.z + v2.z);
  }

  private CVector3 Subtract(CVector3 v1, CVector3 v2) {
    return new CVector3(v1.x - v2.x, v1.y - v2.y, v1.z - v2.z);
  }

  private CVector3 Multiply(CVector3 v1, float m) {
    return new CVector3(v1.x * m, v1.y * m, v1.z * m);
  }

  private CVector3 Divide(CVector3 v1, float m) {
    return new CVector3(v1.x / m, v1.y / m, v1.z / m);
  }
}
```

CFRUSTUM *Diese Klasse stammt auch aus einem Tutorial von ›gametutorials‹. Sie stellt Methoden zur Verfügung, mit denen sich herausfinden lässt, ob ein Objekt im sichtbaren bereich der Kamera liegt und ob es nicht gezeichnet werden muss.*

```
//
//
// simthing.CFrustum
//
// from gametutorials.com
//
// *GT rule make sure to check their tutorials
//
//

//**********************************************************************//
//                                    //
//    - “Talk to me like I’m a 3 year old!” Programming Lessons -     //
//                                                                    //
//    $Author:    DigiBen    digiben@gametutorials.com       //
//                                    //
//    $Program:    Frustum Culling                  //
//                                    //
//    $Description:  Demonstrates checking if shapes are in view     //
//                                    //
//    $Date:      8/28/01                    //
//                                    //
//**********************************************************************//
/*
 Java Port: Ron Sullivan
 Date: 26 Aug 02
 Original C++ version available at : www.GameTutorials.com
 */

package simthing;

import org.lwjgl.opengl.GL;

class CFrustum {

    // We create an enum of the sides so we don’t have to call each side 0 or 1.
    // This way it makes it more understandable and readable when dealing with frustum sides.
    static final int RIGHT = 0; // The RIGHT side of the frustum
    static final int LEFT = 1; // The LEFT   side of the frustum
    static final int BOTTOM = 2; // The BOTTOM side of the frustum
    static final int TOP = 3; // The TOP side of the frustum
    static final int BACK = 4; // The BACK  side of the frustum
    static final int FRONT = 5; // The FRONT side of the frustum

    // Like above, instead of saying a number for the ABC and D of the plane, we
    // want to be more descriptive.
    static final int A = 0; // The X value of the plane’s normal
    static final int B = 1; // The Y value of the plane’s normal
    static final int C = 2; // The Z value of the plane’s normal
    static final int D = 3; // The distance the plane is from the origin

    // This holds the A B C and D values for each side of our frustum.
    float[][] m_Frustum = new float[6][4];

    ////////////////////////////////// NORMALIZE PLANE \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*
    /////
    /////  This normalizes a plane (A side) from a given frustum.
    /////
    ////////////////////////////////// NORMALIZE PLANE \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*

    void NormalizePlane(float[][] frustum, int side) {
        // Here we calculate the magnitude of the normal to the plane (point A B C)
        // Remember that (A, B, C) is that same thing as the normal’s (X, Y, Z).
        // To calculate magnitude you use the equation:  magnitude = sqrt( x^2 + y^2 + z^2)
        float magnitude = (float) Math.sqrt(frustum[side][A] * frustum[side][A] +
                                            frustum[side][B] * frustum[side][B] +
                                            frustum[side][C] * frustum[side][C]);

        // Then we divide the plane’s values by it’s magnitude.
        // This makes it easier to work with.
        frustum[side][A] /= magnitude;
        frustum[side][B] /= magnitude;
        frustum[side][C] /= magnitude;
        frustum[side][D] /= magnitude;
    }

    ////////////////////////////////// CALCULATE FRUSTUM \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*
    /////
    /////  This extracts our frustum from the projection and modelview matrix.
    /////
    ////////////////////////////////// CALCULATE FRUSTUM \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*

    void CalculateFrustum(GL gl) {
        float[] proj = new float[16]; // This will hold our projection matrix
```

```
float[] modl = new float[16]; // This will hold our modelview matrix
float[] clip = new float[16]; // This will hold the clipping planes

// glgetFloatv() is used to extract information about our OpenGL world.
// Below, we pass in PROJECTION_MATRIX to abstract our projection matrix.
// It then stores the matrix into an array of [16].
int projInt = -1;
//gl.getFloatv(gl.PROJECTION_MATRIX, projInt);

// By passing in MODELVIEW_MATRIX, we can abstract our model view matrix.
// This also stores it in an array of [16].
int modlInt = -1;
//gl.getFloatv(gl.MODELVIEW_MATRIX, modlInt);

// Now that we have our modelview and projection matrix, if we combine these 2 matrices,
// it will give us our clipping planes.  To combine 2 matrices, we multiply them.

clip[0] = modl[0] * proj[0] + modl[1] * proj[4] + modl[2] * proj[8] +
    modl[3] * proj[12];
clip[1] = modl[0] * proj[1] + modl[1] * proj[5] + modl[2] * proj[9] +
    modl[3] * proj[13];
clip[2] = modl[0] * proj[2] + modl[1] * proj[6] + modl[2] * proj[10] +
    modl[3] * proj[14];
clip[3] = modl[0] * proj[3] + modl[1] * proj[7] + modl[2] * proj[11] +
    modl[3] * proj[15];

clip[4] = modl[4] * proj[0] + modl[5] * proj[4] + modl[6] * proj[8] +
    modl[7] * proj[12];
clip[5] = modl[4] * proj[1] + modl[5] * proj[5] + modl[6] * proj[9] +
    modl[7] * proj[13];
clip[6] = modl[4] * proj[2] + modl[5] * proj[6] + modl[6] * proj[10] +
    modl[7] * proj[14];
clip[7] = modl[4] * proj[3] + modl[5] * proj[7] + modl[6] * proj[11] +
    modl[7] * proj[15];

clip[8] = modl[8] * proj[0] + modl[9] * proj[4] + modl[10] * proj[8] +
    modl[11] * proj[12];
clip[9] = modl[8] * proj[1] + modl[9] * proj[5] + modl[10] * proj[9] +
    modl[11] * proj[13];
clip[10] = modl[8] * proj[2] + modl[9] * proj[6] + modl[10] * proj[10] +
    modl[11] * proj[14];
clip[11] = modl[8] * proj[3] + modl[9] * proj[7] + modl[10] * proj[11] +
    modl[11] * proj[15];

clip[12] = modl[12] * proj[0] + modl[13] * proj[4] + modl[14] * proj[8] +
    modl[15] * proj[12];
clip[13] = modl[12] * proj[1] + modl[13] * proj[5] + modl[14] * proj[9] +
    modl[15] * proj[13];
clip[14] = modl[12] * proj[2] + modl[13] * proj[6] + modl[14] * proj[10] +
    modl[15] * proj[14];
clip[15] = modl[12] * proj[3] + modl[13] * proj[7] + modl[14] * proj[11] +
    modl[15] * proj[15];

// Now we actually want to get the sides of the frustum.  To do this we take
// the clipping planes we received above and extract the sides from them.

// This will extract the RIGHT side of the frustum
m_Frustum[RIGHT][A] = clip[3] - clip[0];
m_Frustum[RIGHT][B] = clip[7] - clip[4];
m_Frustum[RIGHT][C] = clip[11] - clip[8];
m_Frustum[RIGHT][D] = clip[15] - clip[12];

// Now that we have a normal (A,B,C) and a distance (D) to the plane,
// we want to normalize that normal and distance.

// Normalize the RIGHT side
NormalizePlane(m_Frustum, RIGHT);

// This will extract the LEFT side of the frustum
m_Frustum[LEFT][A] = clip[3] + clip[0];
m_Frustum[LEFT][B] = clip[7] + clip[4];
m_Frustum[LEFT][C] = clip[11] + clip[8];
m_Frustum[LEFT][D] = clip[15] + clip[12];

// Normalize the LEFT side
NormalizePlane(m_Frustum, LEFT);

// This will extract the BOTTOM side of the frustum
m_Frustum[BOTTOM][A] = clip[3] + clip[1];
m_Frustum[BOTTOM][B] = clip[7] + clip[5];
m_Frustum[BOTTOM][C] = clip[11] + clip[9];
m_Frustum[BOTTOM][D] = clip[15] + clip[13];

// Normalize the BOTTOM side
    NormalizePlane(m_Frustum, BOTTOM);

    // This will extract the TOP side of the frustum
    m_Frustum[TOP][A] = clip[3] - clip[1];
    m_Frustum[TOP][B] = clip[7] - clip[5];
    m_Frustum[TOP][C] = clip[11] - clip[9];
    m_Frustum[TOP][D] = clip[15] - clip[13];

    // Normalize the TOP side
    NormalizePlane(m_Frustum, TOP);

    // This will extract the BACK side of the frustum
    m_Frustum[BACK][A] = clip[3] - clip[2];
    m_Frustum[BACK][B] = clip[7] - clip[6];
    m_Frustum[BACK][C] = clip[11] - clip[10];
    m_Frustum[BACK][D] = clip[15] - clip[14];

    // Normalize the BACK side
    NormalizePlane(m_Frustum, BACK);

    // This will extract the FRONT side of the frustum
    m_Frustum[FRONT][A] = clip[3] + clip[2];
    m_Frustum[FRONT][B] = clip[7] + clip[6];
    m_Frustum[FRONT][C] = clip[11] + clip[10];
    m_Frustum[FRONT][D] = clip[15] + clip[14];

    // Normalize the FRONT side
    NormalizePlane(m_Frustum, FRONT);
}

// The code below will allow us to make checks within the frustum.  For example,
// if we want to see if a point, a sphere, or a cube lies inside of the frustum.
// Because all of our planes point INWARDS (The normals are all pointing inside the frustum)
// we then can assume that if a point is in FRONT of all of the planes, it's inside.

////////////////////////////////// POINT IN FRUSTUM \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*
/////
/////  This determines if a point is inside of the frustum
/////
////////////////////////////////// POINT IN FRUSTUM \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*

boolean PointInFrustum(float x, float y, float z) {
    // If you remember the plane equation (A*x + B*y + C*z + D = 0), then the rest
    // of this code should be quite obvious and easy to figure out yourself.
    // In case don't know the plane equation, it might be a good idea to look
    // at our Plane Collision tutorial at www.GameTutorials.com in OpenGL Tutorials.
    // I will briefly go over it here.  (A,B,C) is the (X,Y,Z) of the normal to the plane.
    // They are the same thing... but just called ABC because you don't want to say:
    // (x*x + y*y + z*z + d = 0).  That would be wrong, so they substitute them.
    // the (x, y, z) in the equation is the point that you are testing.  The D is
    // The distance the plane is from the origin.  The equation ends with "= 0" because
    // that is true when the point (x, y, z) is ON the plane.  When the point is NOT on
    // the plane, it is either a negative number (the point is behind the plane) or a
    // positive number (the point is in front of the plane).  We want to check if the point
    // is in front of the plane, so all we have to do is go through each point and make
    // sure the plane equation goes out to a positive number on each side of the frustum.
    // The result (be it positive or negative) is the distance the point is front the plane.

    // Go through all the sides of the frustum
    for (int i = 0; i < 6; i++) {
        // Calculate the plane equation and check if the point is behind a side of the frustum
        if (m_Frustum[i][A] * x + m_Frustum[i][B] * y + m_Frustum[i][C] * z +
            m_Frustum[i][D] <= 0) {
            // The point was behind a side, so it ISN'T in the frustum
            return false;
        }
    }

    // The point was inside of the frustum (In front of ALL the sides of the frustum)
    return true;
}

////////////////////////////////// SPHERE IN FRUSTUM \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*
/////
/////  This determines if a sphere is inside of our frustum by it's center and radius.
/////
////////////////////////////////// SPHERE IN FRUSTUM \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*

boolean SphereInFrustum(float x, float y, float z, float radius) {
    // Now this function is almost identical to the PointInFrustum(), except we
    // now have to deal with a radius around the point.  The point is the center of
    // the radius.  So, the point might be outside of the frustum, but it doesn't
    // mean that the rest of the sphere is.  It could be half and half.  So instead of
    // checking if it's less than 0, we need to add on the radius to that.  Say the
```

```
    // equation produced -2, which means the center of the sphere is the distance of
    // 2 behind the plane.  Well, what if the radius was 5?  The sphere is still inside,
    // so we would say, if(-2 < -5) then we are outside.  In that case it's false,
    // so we are inside of the frustum, but a distance of 3.  This is reflected below.

    // Go through all the sides of the frustum
    for (int i = 0; i < 6; i++) {
        // If the center of the sphere is farther away from the plane than the radius
        if (m_Frustum[i][A] * x + m_Frustum[i][B] * y + m_Frustum[i][C] * z +
            m_Frustum[i][D] <= -radius) {
            // The distance was greater than the radius so the sphere is outside of the frustum
            return false;
        }
    }

    // The sphere was inside of the frustum!
    return true;
}

/////////////////////////////// CUBE IN FRUSTUM \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

/////
/////  This determines if a cube is in or around our frustum by it's center and 1/2 it's length
/////
/////////////////////////////// CUBE IN FRUSTUM \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

boolean CubeInFrustum(float x, float y, float z, float size) {
    // This test is a bit more work, but not too much more complicated.
    // Basically, what is going on is, that we are given the center of the cube,
    // and half the length.  Think of it like a radius.  Then we checking each point
    // in the cube and seeing if it is inside the frustum.  If a point is found in front
    // of a side, then we skip to the next side.  If we get to a plane that does NOT have
    // a point in front of it, then it will return false.

    // *Note* - This will sometimes say that a cube is inside the frustum when it isn't.
    // This happens when all the corners of the bounding box are not behind any one plane.
    // This is rare and shouldn't effect the overall rendering speed.

    for (int i = 0; i < 6; i++) {
        if (m_Frustum[i][A] * (x - size) + m_Frustum[i][B] * (y - size) +
            m_Frustum[i][C] * (z - size) + m_Frustum[i][D] > 0)
            continue;
        if (m_Frustum[i][A] * (x + size) + m_Frustum[i][B] * (y - size) +
            m_Frustum[i][C] * (z - size) + m_Frustum[i][D] > 0)
            continue;
        if (m_Frustum[i][A] * (x - size) + m_Frustum[i][B] * (y + size) +
            m_Frustum[i][C] * (z - size) + m_Frustum[i][D] > 0)
            continue;
        if (m_Frustum[i][A] * (x + size) + m_Frustum[i][B] * (y + size) +
            m_Frustum[i][C] * (z - size) + m_Frustum[i][D] > 0)
            continue;
        if (m_Frustum[i][A] * (x - size) + m_Frustum[i][B] * (y - size) +
            m_Frustum[i][C] * (z + size) + m_Frustum[i][D] > 0)
            continue;
        if (m_Frustum[i][A] * (x + size) + m_Frustum[i][B] * (y - size) +
            m_Frustum[i][C] * (z + size) + m_Frustum[i][D] > 0)
            continue;
        if (m_Frustum[i][A] * (x - size) + m_Frustum[i][B] * (y + size) +
            m_Frustum[i][C] * (z + size) + m_Frustum[i][D] > 0)
            continue;
        if (m_Frustum[i][A] * (x + size) + m_Frustum[i][B] * (y + size) +
            m_Frustum[i][C] * (z + size) + m_Frustum[i][D] > 0)
            continue;

        // If we get here, it isn't in the frustum
        //return false;
        return true;
    }

    return true;
  }
}
```

TEXTURELOADER *Diese Klasse lädt Texturen in den OpenGL speicher.*

```
//
//
// TextureLoader
//
// some methods or lines were taken from
// NeHe tutorials
// and gametutorials.com
//
// * NeHe and GT, both rule make sure to check their tutorials
//
//

package simthing;


import java.io.*;
import java.awt.geom.*;
import java.awt.Image;
import java.awt.Graphics2D;
import java.awt.image.*;
import java.nio.*;

import org.lwjgl.*;
import org.lwjgl.opengl.*;

// A Texture loader

public final class TextureLoader {

    private final static int[] loadTexture(String path) {
        Image image = (new javax.swing.ImageIcon(path)).getImage();
        // Exctract The Image
        BufferedImage tex = new BufferedImage(image.getWidth(null), image.getHeight(null),
                                              BufferedImage.TYPE_3BYTE_BGR);
        Graphics2D g = (Graphics2D) tex.getGraphics();
        g.drawImage(image, null, null);
        g.dispose();
        // Flip Image
        AffineTransform tx = AffineTransform.getScaleInstance(1, -1);
        tx.translate(0, -image.getHeight(null));
        AffineTransformOp op = new AffineTransformOp(tx,
                                                     AffineTransformOp.
                                                     TYPE_NEAREST_NEIGHBOR);
        tex = op.filter(tex, null);
        // Put Image In Memory
        ByteBuffer scratch = ByteBuffer.allocateDirect(4 * tex.getWidth() *
                                                       tex.getHeight());
        int scratchAddress = Sys.getDirectBufferAddress(scratch);
        byte data[] = (byte[]) tex.getRaster().getDataElements(0, 0, tex.getWidth(),
                                                               tex.getHeight(), null);
        scratch.clear();
        scratch.put(data);
        // Create A IntBuffer For Image Address In Memory
        IntBuffer buf = ByteBuffer.allocateDirect(12).order(ByteOrder.nativeOrder()).
            asIntBuffer();
        int bufPtr = Sys.getDirectBufferAddress(buf); // Get Image Address
        gl.genTextures(3, bufPtr); // Create Texture In OpenGL
        // Create Nearest Filtered Texture
        gl.bindTexture(GL.TEXTURE_2D, buf.get(0));
        gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MIN_FILTER, GL.NEAREST);
        gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MAG_FILTER, GL.NEAREST);
        gl.texImage2D(GL.TEXTURE_2D, 0, GL.RGB, tex.getWidth(), tex.getHeight(), 0,
                      GL.RGB, GL.UNSIGNED_BYTE, scratchAddress);
        // Create Linear Filtered Texture
        gl.bindTexture(GL.TEXTURE_2D, buf.get(1));
        gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MIN_FILTER, GL.LINEAR);
        gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MAG_FILTER, GL.LINEAR);
        gl.texImage2D(GL.TEXTURE_2D, 0, GL.RGB, tex.getWidth(), tex.getHeight(), 0,
                      GL.RGB, GL.UNSIGNED_BYTE, scratchAddress);
        // Create MipMapped Texture
        gl.bindTexture(GL.TEXTURE_2D, buf.get(2));
        gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MIN_FILTER, GL.LINEAR);
        gl.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MAG_FILTER,
                         GL.LINEAR_MIPMAP_NEAREST);
        glu.build2DMipmaps(GL.TEXTURE_2D, 3, tex.getWidth(), tex.getHeight(),
                           GL.RGB, GL.UNSIGNED_BYTE, scratchAddress);
        return new int[] {
            buf.get(0), buf.get(1), buf.get(2)}; // Return Image Addresses In Memory
    }
}
```

VECTOR2F *Diese Klasse stellt ein Vektor Objekt zur Verfügung. Sie basiert auf der Klasse ›vecmath‹ die ein Teil der Java3D API ist, allerdings ist sie für die Verwendung von 2 komponenten Vektoren geschrieben. Es werden teilweise Begriffe aus der 3DVektorMathematik benutzt, die so nicht allgemein nicht benutzt werden, aber meinem persönlichen Verständnis dienen.*

```
//
//
// Vector2f
//
//

package simthing;

import java.io.Serializable;

public class Vector2f
extends Tuple2f
implements Serializable {

    /* ---------------------------------------------------------------- */
    // constructors
    /* ---------------------------------------------------------------- */
    public Vector2f(float x, float y) {
        super(x, y);
    }

    public Vector2f(float v[]) {
        super(v);
    }

    public Vector2f(Vector2f v1) {
        super(v1);
    }

    public Vector2f(Tuple2f t1) {
        super(t1);
    }

    public Vector2f() {
        super();
    }

    /* ---------------------------------------------------------------- */
    // methods
    /* ---------------------------------------------------------------- */
    // lengthSquared
    /* ---------------------------------- */
    public final float lengthSquared() {
        return x * x + y * y;
    }

    /* ---------------------------------- */
    // length
    /* ---------------------------------- */
    public final float length() {
        return (float) Math.sqrt(lengthSquared());
    }

    /* ---------------------------------- */
    // cross ( rather the 2D equivalent )
    /* ---------------------------------- */
    public final void cross(Vector2f v1) {
        set(
            -v1.y,
            v1.x
            );
    }

    /* ---------------------------------- */
    // cross ( rather the 2D equivalent )
    /* ---------------------------------- */
    public final void cross() {
        float n = x;
        x = -y;
        y = n;
    }

    /* ---------------------------------- */
    // dot ( rather the 2D equivalent )
    /* ---------------------------------- */
    public final float dot(Vector2f v1) {
        return x * v1.x + y * v1.y;
    }

    /* ---------------------------------- */
    // normalize
    /* ---------------------------------- */
    public final void normalize(Vector2f v1) {
        set(v1);
        normalize();
    }

    public final void normalize() {
        double d = length();
        if (d != 0.0f) {
            x /= d;
            y /= d;
        }
    }

    /* ---------------------------------- */
    // setLength
    /* ---------------------------------- */
    public final void setLength(float s, Vector2f v1) {
        set(v1);
        setLength(s);
    }

    public final void setLength(float s) {
        double d = this.length();
        x /= d;
        y /= d;
        x *= s;
        y *= s;
    }

    /* ---------------------------------- */
    // truncLength
    /* ---------------------------------- */
    public final void truncLength(float max, Vector2f v1) {
        set(v1);
        truncLength(max);
    }

    public final void truncLength(float max) {
        float l = length();
        if (l > max && l != 0) {
            scale(max / l);
        }
    }

    /* ---------------------------------- */
    // minLength
    /* ---------------------------------- */
    public final void minLength(float min, Vector2f v1) {
        set(v1);
        truncLength(min);
    }

    public final void minLength(float min) {
        float l = length();
        if (l < min && l != 0) {
            scale(min / l);
        }
    }

    /* ---------------------------------- */
    // angle ( in radians, range [0,PI] )
    /* ---------------------------------- */
    public final float angle(Vector2f v1) {
        double cross = x * v1.y - y * v1.x;
        return (float) Math.abs(Math.atan2(cross, dot(v1)));
    }

    /* ---------------------------------- */
    // angle ( in radians, range [0,PI] )
    /* ---------------------------------- */
    public final float angleReal(Vector2f v1) {
        double cross = x * v1.y - y * v1.x;
        return (float) Math.atan2(cross, dot(v1));
    }
}
```

```
//
//
// Tuple2f /// written by ???
//
//

package simthing;

import java.io.Serializable;

public abstract class Tuple2f
implements Serializable {

    /* ---------------------------------------------------------------- */
    // variables
    /* ---------------------------------------------------------------- */

    public float x;
    public float y;

    /* ---------------------------------------------------------------- */
    // constructors
    /* ---------------------------------------------------------------- */

    public Tuple2f(float x, float y) {
        this.x = x;
        this.y = y;
    }

    public Tuple2f(float t[]) {
        this.x = t[0];
        this.y = t[1];
    }

    public Tuple2f(Tuple2f t1) {
        x = t1.x;
        y = t1.y;
    }

    public Tuple2f() {
        x = 0.0f;
        y = 0.0f;
    }

    /* ---------------------------------------------------------------- */
    // methods
    /* ---------------------------------------------------------------- */

    /* --------------------------------- */
    // set
    /* --------------------------------- */
    public final void set(float x, float y) {
        this.x = x;
        this.y = y;
    }

    public final void set(float t[]) {
        x = t[0];
        y = t[1];
    }

    public final void set(Tuple2f t1) {
        x = t1.x;
        y = t1.y;
    }

    /* --------------------------------- */
    // get
    /* --------------------------------- */
    public final void get(float t[]) {
        t[0] = x;
        t[1] = y;
    }

    public final void get(Tuple2f t) {
        t.x = x;
        t.y = y;
    }

    /* --------------------------------- */
    // add
    /* --------------------------------- */
    public final void add(Tuple2f t1, Tuple2f t2) {
        x = t1.x + t2.x;
        y = t1.y + t2.y;
    }

    public final void add(Tuple2f t1) {
        x += t1.x;
        y += t1.y;
    }

    /* --------------------------------- */
    // sub
    /* --------------------------------- */
    public final void sub(Tuple2f t1, Tuple2f t2) {
        x = t1.x - t2.x;
        y = t1.y - t2.y;
    }

    public final void sub(Tuple2f t1) {
        x -= t1.x;
        y -= t1.y;
    }

    /* --------------------------------- */
    // negate
    /* --------------------------------- */
    public final void negate(Tuple2f t1) {
        x = -t1.x;
        y = -t1.y;
    }

    public final void negate() {
        x = -x;
        y = -y;
    }

    /* --------------------------------- */
    // scale
    /* --------------------------------- */
    public final void scale(float s, Tuple2f t1) {
        x = s * t1.x;
        y = s * t1.y;
    }

    public final void scale(float s) {
        x *= s;
        y *= s;
    }

    /* --------------------------------- */
    // scaleAdd
    /* --------------------------------- */
    public final void scaleAdd(float s, Tuple2f t1, Tuple2f t2) {
        x = s * t1.x + t2.x;
        y = s * t1.y + t2.y;
    }

    public final void scaleAdd(float s, Tuple2f t1) {
        x = s * x + t1.x;
        y = s * y + t1.y;
    }

    /* --------------------------------- */
    // equals
    /* --------------------------------- */
    public boolean equals(Tuple2f t1) {
        return t1 != null && x == t1.x && y == t1.y;
    }

    /* --------------------------------- */
    // toString
    /* --------------------------------- */
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
```

```
    /* --------------------------------- */
    // clamp
    /* --------------------------------- */
    public final void clamp(float min, float max, Tuple2f t) {
        set(t);
        clamp(min, max);
    }

    public final void clamp(float min, float max) {
        clampMin(min);
        clampMax(max);
    }

    public final void clampMin(float min, Tuple2f t) {
        set(t);
        clampMin(min);
    }

    public final void clampMin(float min) {
        if (x < min)
            x = min;
        if (y < min)
            y = min;
    }

    public final void clampMax(float max, Tuple2f t) {
        set(t);
        clampMax(max);
    }

    public final void clampMax(float max) {
        if (x > max)
            x = max;
        if (y > max)
            y = max;
    }

    /* --------------------------------- */
    // absolute
    /* --------------------------------- */
    public final void absolute(Tuple2f t) {
        set(t);
        absolute();
    }

    public final void absolute() {
        if (x < 0.0)
            x = -x;
        if (y < 0.0)
            y = -y;
    }

    /* --------------------------------- */
    // interpolate
    /* --------------------------------- */
    public final void interpolate(Tuple2f t1, Tuple2f t2, float alpha) {
        set(t1);
        interpolate(t2, alpha);
    }

    public final void interpolate(Tuple2f t1, float alpha) {
        float beta = 1 - alpha;
        x = beta * x + alpha * t1.x;
        y = beta * y + alpha * t1.y;
    }
}
```

GROSSEN DANK AN

Linus Heins
Annika Heins
Meine Mutter

Eva Engler
Monika Hoinkis
Jakob Lehr
Falko Oldenburg
Thomas Tietjen

Professor Joachim Sauter

Apple Computer