Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht

Sven Tennstedt
Julia Pressburger

Emotional Tent
Ambient Awareness

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sven Tennstedt
Julia Pressburger

Emotional Tent

Ambient Awareness

Projektbericht eingereicht im Rahmen des Masterprojektes
im Studiengang Master of Science - Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Kai von Luck
Zweitgutachter : Gunter Klemke

Abgegeben am 9. März 2009

Inhaltsverzeichnis

1. Einführung

(Sven Tennstedt und Julia Pressburger)

Das Projekt Emotional Tent, Ambient Awareness, des WS 2008/2009 war ein interdisziplinäres Pilotprojekt der Hochschule für Angewandte Wissenschaften Hamburg zwischen den Fakultäten *Design, Medien und Information* und *Technik und Informatik*.

Die Aufgabe war es eine interaktive Ausstellung zu realisieren, in der der Besucher auf neue und ungewohnte Weise mit der Technik interagieren kann.

Für dieses Projekt trafen zwei Gruppen von Menschen aufeinander, Künstler und Techniker. Beide Gruppen hatten wenig oder keine Erfahrung in der Zusammenarbeit mit Personen der jeweils anderen Gruppe.

Eine wissenschaftliche Untersuchung dieser Art der Zusammenstellung in Projektgruppen findet sich in „An in-depth case study of art-technology collaboration“ (Zhang und Candy, 2007). In der besonders die Kommunikatiossituationen während des Projektes untersucht hat und feststellte, dass während eines Gesprächs zwischen Technikern und Künstlern über technische Aspekte diese am Computer gleich demonstriert wurden oder Zeichnungen angefertigt wurden um das Gesagte zu verdeutlichen.

Das Projekt *Ambient Awareness* ist im Rahmen des Forschungsschwerpunkts

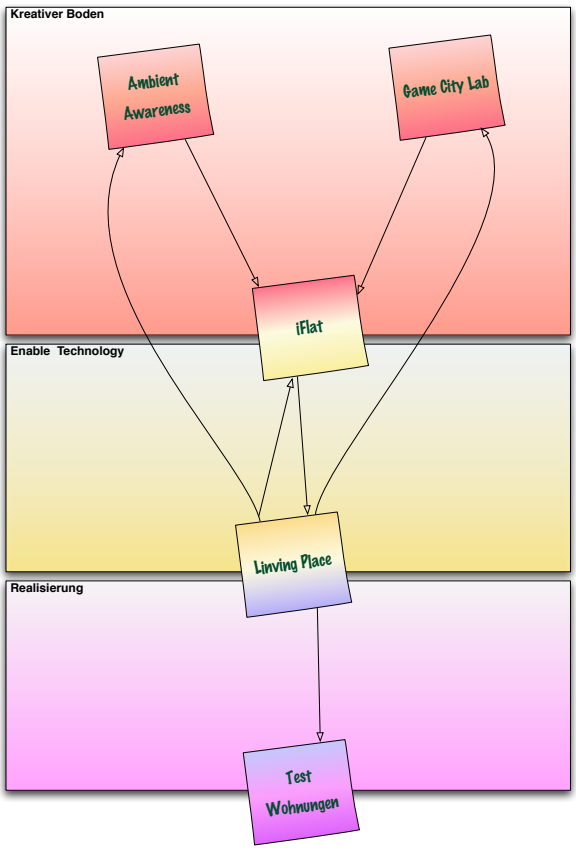


Abbildung 1.1.: **Ambient Awareness** im Forschungsschwerpunkt Ambient Assistant Living

Ambient Intelligence in die Grundlagenforschung einzuordnen. Es soll Ideenschmiede und Anwendungstest zugleich sein.

1.1. Motivation

Die Motivtaion für das Projekt war anfangs, dass es ein großes Interesse gab an einem interdisziplinären Projekt in dieser Größenordnung mitzuwirken.

Des Weiteren ist Kunst als Ausdrucksform spannend. Zudem gab es die Möglichkeit Zugang zu Menschen zu haben, die kein hohes technisches Interesse mitbrachten oder Technik zuerst einmal skeptisch gegenüber standen.

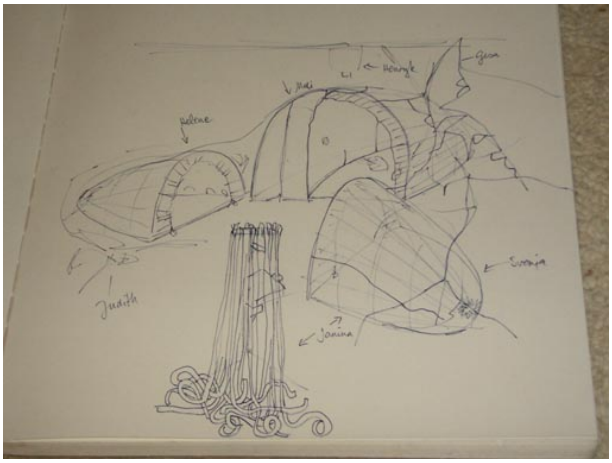


Abbildung 1.2.: **Tent - Zeichnung** (ambientawareness, 2009)

Und letztendlich Informatik als *enabeling technology*, für erlebbare und anfassbare Kunst - eine große Chance und eine tolle Erfahrung auf allen Seiten:

Auf einer Seite der Künstler, der sich auf seine kreative Arbeit konzentrieren kann und sich nicht mit Technik herumärgern muss. Auf der anderen Seite der Besucher, der eine Ausstellung betritt, die auf ihn reagiert und ihn dazu animiert das Ausgestellte aktiv zu erleben und mitzugestalten. Mittendrin Informatiker, mit der Chance kreative Ideen, der Köpfe die sich mit damit auskennen, mit zu realisieren, *und* die Chance in einem eingegrenzten Szenario neue Technologien und Interaktionstechniken auszuprobieren und dies gleichzeitig als Usablity-Test zu nutzen.

1.2. Interactive Design

Interactive Design (siehe auch (Pressburger, 2009a)) gehört in den Bereich User-Centered-Design. Für das Projekt stellte sich die Frage, wie man den Ausstellungsaufbau gestallten kann, so dass:

- der Besucher im Mittelpunkt steht.
- der Besucher Lust bekommt mit der Ausstellung zu interagieren.

- dem Besucher ohne Erklärung deutlich wird, wie er mit der Austellung zu interagieren hat, welches die Interaktionsgesten sind.
- die Technik dabei im Hintergrund bleibt.

(Adamczyk u. a., 2007)

Zur Realisierung der geforderten Interaktionsmodelle wurden Technologien aus dem Bereich „Seamless Interaction“ (siehe Kapitel 1.3) untersucht.

Der User-Centered-Aufbau der Ausstellung wurde zum großen Teil den Studenten aus dem Department Design überlassen. Bei den Besprechungen wollten die Informatiker darauf achten, dass bei der Gestaltung der Ausstellung die Funktion der eingesetzten Technik nicht gestört werden würde. Die Technik sollte zwar in den Hintergrund treten, „seamless“ sein, doch würde z. B. ein Beamer eine freie Fläche zum Projizieren brauchen.

1.3. Seamless Interaction

„Seamless Interaction“ bedeutet, im Zusammenhang mit einer Ausstellung von Interaktiver Kunst, dass die Technik in den Hintergrund tritt und nicht offensichtlich sichtbar und spürbar ist (für Interaktionstechniken im Digital-Art-Bereich siehe auch (Pressburger, 2009a).

Für das Projekt wurden unterschiedliche Interaktionstechniken betrachtet. Bei den, in diesem Bericht vorgestellten, Einzelprojekten entschied man sich für folgende:

- Indoor-Location-System (Kapitel 3)
- Kamera zur Bewegungserkennung (Kapitel 6)

Das Projekt WeFollowYou! (in Kapitel 2) setzte das Indoor-Location-System ein. Die Projekte *„Little Vintage Garden“* (in Kapitel 5) und *„Skinsight“* (in Kapitel 4) setzten eine einfache USB-Kamera für die Bewegungserkennung ein.

2. WeFollowYou!

(Sven Tennstedt)

„*von Henryk Wollik (Kommunikationsdesign/Processing) & Sven Tennstedt (Informatik/Java)*

Info: Lebewesen sind Konglomerate einer Vielzahl von Organismen. WeFollowYou! zeigt einen vergrößerten Ausschnitt eines für Menschen physisch unerfahrbaren Mikro-Universums. Das virtuelle Habitat schafft eine Verbindung beider Welten, lässt somit Mensch und Kleinstkreatur auf selber Fläche koexistieren und auf einander reagieren.

Technik: Java/ Processing (MIT), Indoor Location System, Projektion

WeFollowYou! ist allein auf code-basis entstanden.“ (ambientawareness, 2009)

Dieses Kapitel beschäftigt sich mit dem Projekt WeFollowYou!, an welchem Tennstedt, ein Master–Informatik–Student des Departments Technik und Informatik, und Wollik, ein Kommunikationsdesign–Student des Departments Design, Medien und Information, gearbeitet haben.

Abbildung 2.1.: **WeFollowYou!** AmbientAwareness.org

Als erstes wird das Ziel von WeFollowYou! erläutert (Abschnitt 2.1). Anschließend wird die Technologie vorgestellt, mit der das Projekt umgesetzt wurde (Abschnitt 2.2, S. 9). Dann folgt die Beschreibung der eigentlichen Realisierung (Abschnitt 2.3, S. 11). Am Schluss werden die Ergebnisse ausgewertet (Abschnitt 2.4, S. 13).

2.1. Ziel

Das Projektziel von WeFollowYou! war es einen Schwarm von Mikroorganismen zu erschaffen, der auf den Besucher reagiert. Die Mikroorganismen sollten unabhängig vom Besucher ein Eigenleben führen und Gemeinschaften bilden. Daher wurde geplant, dass es verschiedene Arten von Organismen geben soll, die miteinander auf verschiedene Weise interagieren. Die Organismen wurden hierarchisch in Mütter, Kinder und Mikroben eingeteilt.

Abbildung 2.2.: Mutter mit Kindern und Mikroben

Die Mütter sollten selbstständig umherstreifen und Attraktionen für die anderen Organismen darstellen. Zudem wurde ein Hort geplant, in dem sich die Kinder und Mikroben, die über einen begrenzten Energievorrat verfügen, ausruhen können, wenn sie erschöpft sind. Die Besucher sollten den Mütter entsprechend für die kleineren Organismen als Attraktionen dienen.

Vorraussetzung für die Interaktion mit dem Besucher war, dass WeFollowYou! auf den Boden projiziert wird, damit die Besucher die Welt der Mikroorganismen betreten konnten. Auf diese Weise sollte eine einfache und intuitive Interaktion mit der Welt ermöglicht werden.

2.2. Technologie

2.2.1. Software

Für die Entwicklung von WeFollowYou! wurde Processing (Reas und Fry) eingesetzt, das im Media–Lab der MIT entwickelt wurde, um Medienkünstlern einen leichten Zugang in die Grafikprogrammierung zu ermöglichen. In der Processing–Umgebung wird in Java programmiert, die Komplexität von Java wird allerdings weitestgehend vor dem Anwender verborgen. Ein Medienkünstler kann mit einfachen Befehlen direkt auf den Bildschirm zeichnen (wie in „Digital Art Design“ (Pressburger, 2009a) zu lesen).

Das folgende Beispiel erstellt eine Zeichenfläche von der Größe *200 x 200* Pixel, färbt den Hintergrund schwarz und zeichnet ein zentriertes graues Rechteck mit einem *10* Pixel Rand:

```
void setup() {
  size(200, 200);
}

void draw() {
  background(0);
  fill(40);
  rect(10, 10, width-20, height-20);
}
```

An diesem Beispiel ist deutlich zu sehen, wieviel von der Java–Syntax versteckt wird. Als erstes fällt auf, dass keine Klassen importiert werden müssen und die eigentliche Klassendefinition weggelassen werden kann. Zudem brauchen keine *Sichtbarkeitsmodifizierer* angegeben werden. Aus dem Beispiel oben wird nach einem Export folgender Code:

```
import processing.core.*;
import processing.xml.*;

// weitere imports...

public class sketch_mar05a extends PApplet {

  public void setup() {
    size(200, 200);
  }

  public void draw() {
    background(0);
    fill(40);
    rect(10, 10, width-20, height-20);
  }

  static public void main(String args[]) {
    PApplet.main(new String[] { "--bgcolor=#FFFFFF", "sketch_mar05a" });
  }
}
```

Die Processing–Anwendung erbt von *PApplet*, dem eigentlichen Herzstück von Processing. Das PApplet ist eine Unterklasse von *Applet*. Es stellt alle Funktionen zum Zeichnen und zur Konfiguration der Processing–Anwendung zur Verfügung.

Processing war für die Verwirklichung der graphischen Komponente in WeFollowYou! die erste Wahl, zum einen, weil Wollik schon Erfahrung darin vorzuweisen hatte zum anderen hätte die Einarbeitung in alternative Frameworks für einen nichtgelernten Programmierer wie Wollik einen zu hohen Aufwand dargestellt, der nicht mit der zur Verfügung stehend Zeit im Verhältnis gestanden hätte. Alternativen wären z. B. *openFrameworks* (siehe (Pressburger,

2009a)), das in C++ programmiert wird, oder die native Programmierung von OpenGL oder Direct3D über C++, C# oder Java.

2.2.2. Hardware

Um die Welt später auf den Boden projizieren zu können, war ein Beamer vorgesehen, der senkrecht von oben auf die Projektionsfläche strahlt. Hierzu war neben einem geeigneten Beamer eine Halterung notwendig, mit der der Beamer senkrecht nach unten zeigend angebracht werden konnte.

Für die Interaktion mit den Besuchern war das System für Echtzeit–Ortung Ubisense vorgesehen, das im Kapitel 3 (S. 15ff) beschrieben wird. Die Wahl fiel hierauf, u. a. weil es verfügbar war und bereits Erfahrungen im iFlat gesammelt werden konnten, und weil das Projekt die Möglichkeit bot, das System in einer großen Installation zu testen.

2.3. Realisierung

2.3.1. Software

Wollik arbeitete die meiste Zeit in der Processing–IDE, da dies für ihn die gewohntere Programmierumgebung darstellte. Die andere Teamhälfte Tennstedt arbeitete hingegen in der Eclipse–IDE, weil hier ein ausgereifter Subversion–Support, sowie Debugging– und Refactoring–Tools zur Verfügung standen.

Diese über weite Strecken getrennte Arbeitsweise erforderte später eine Migration des Programmcodes aus Processing in das Eclipse–Projekt. Zwar war es möglich aus Processing den Programmcode zu exportieren, dennoch mussten einige Anpassung per Hand vorgenommen werden.

Abbildung 2.3 (S. 12) zeigt schematisch die Architektur von WeFollowYou!. Das WeFollowYou!–Processing–Applet stellte die zentrale Einheit dar. Es enthielt u. a. den Controller für die Simulation der Organismen (*SwarmController*). Für jede Entität eines Mikroorganismus gab es zwei Repäsentationen, einmal die Instanz für das Zeichnen des Organismus auf dem Bildschirm (*VisualEntity*), zum anderen die für die Erzeugung des Verhaltens (*SwarmEntity*).

Die Simulation war zeitkontinuierlich und synchron getaktet, d. h. die einzelnen Entitäten wurden bei jedem globalen Simulationsschritt getriggert, damit diese ihren individuellen Simulationsschritt ausführen. Vor jedem globalen Simulationsschritt wurden in einem Quadtree

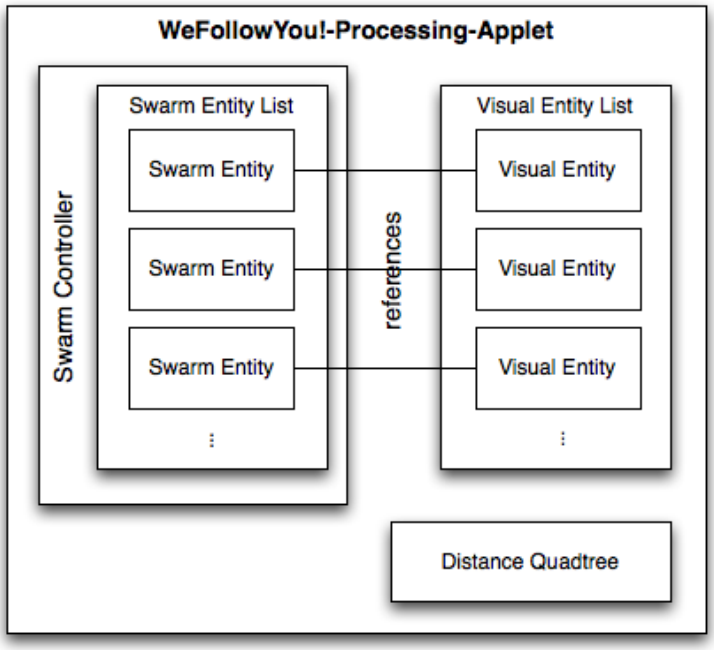


Abbildung 2.3.: Architektur von WeFollowYou!

die Distanzen der Entitäten zueinander gespeichert, damit die einzelnen Entitäten abfragen konnten, welche anderen Entitäten sich in ihrer Nähe befanden.

Exemplarisch sind die Verhaltensregeln der Mütter und Mikroben im Anhang A.2 (S. 36) zu finden.

WeFollowYou! lief bei der Ausstellung auf einem Apple MacPro mit vier Kernen unter Windows XP. Processing konnte allerdings nur mit einem Thread arbeiten. Daher wurde die Leistung der vier Kerne nicht ausgenutzt. Aus diesem Grund war ab ca. 400 Entitäten keine flüssige Simulation mehr möglich. Für die Ausstellung wurde daher die Anzahl auf 360 festgelegt.

2.3.2. Beamer und Aufhängung

In der Aula des Departments Design, Medien und Information gab es zwei Traversen in 4 Meter Höhe. An einer dieser Traversen hing bereits ein Beamer. Dieser konnte jedoch nur in der Horizontalen projizieren. Daher war es nötig einen zusätzlichen Beamer anzubringen, der so befestigt war, dass er vertikal nach unten projizieren konnte.

Nach einigen vergeblichen Versuchen eine passende Halterung über Veranstaltungsausstatter wie Amptown (Amptown, 2009) zu beziehen, wurde entschieden, diese mit Hilfe der Zentralwerkstatt der HAW Hamburg selbst zu bauen. Abbildung 2.4 zeigt den skizzenhafte Bauplan nach dem die Halterung angefertigt. Die Maße beziehen sich auf das Beamermodell Epson EMP 7250, der ursprünglich für dieses Projekt geplant war.

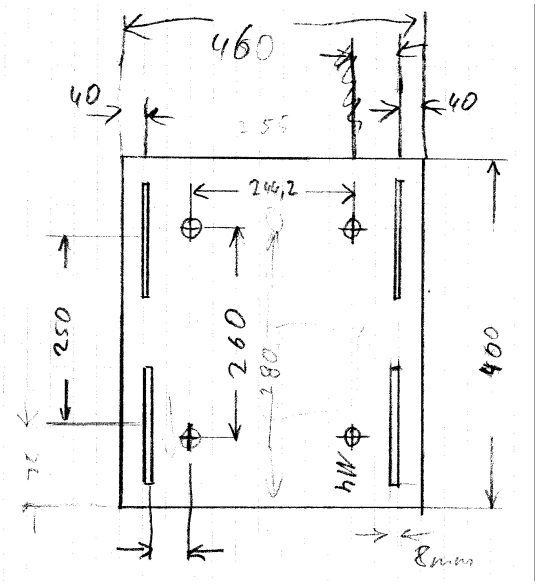


Abbildung 2.4.: technische Skizze für die Konstruktion der Beamerhalterung

Verwendet wurde eine vier Millimeter starke Aluminiumplatte. In diese mussten Schlitze für Schellen und ein rechteckiges Loch für die Beamerlüftung gestanzt werden, sowie Löcher gebohrt werden, um den Beamer an der Platte festzuschrauben.

Der Epson besaß allerdings eine Helligkeit von nur 1300 Ansi–Lumen, was sich in Laufe von Tests als zu dunkel für das Vorhaben herausstellte. Daher wurde ein neuer Beamer der Firma NEC mit ca. 3000 AnsiLumen bestellt. Dieser besaß allerdings andere Maße.

Da der neue Beamer erst vier Tage vor Ausstellungsbeginn eintraf, konnte keine neue Befestigung konstruiert werden, sondern es wurden einige zusätzliche Löcher in die bestehende Platte gebohrt. Trotz dieser Maßnahmen blieb die Stabilität der Konstruktion erhalten.

Der NEC stellte sich als hell genug heraus.

2.4. Fazit und Blick nach vorn

Die Software lief stabil und zeigte das gewünschte Verhalten. Selbst nach mehreren Stunden behielt der Schwarm seine Dynamik und lief in keinen festen Zustand.

Processing bot viele Möglichkeiten, die mit einer Programmierung auf direkter 2D– und 3D–Zeichenebene unter Direct3D oder OpenGL nur mit erheblich mehr Aufwand hätten realisiert werden können. Aus diesem Grund ist Processing für diese Art von Projekten sehr empfehlenswert. Ein großer Schwachpunkt stellte jedoch die Performance dar. Für aufwändigere Animationen ist es daher empfehlenswert Alternativen, wie z. B. *openFrameworks* (Pressburger, 2009a), in Betracht zu ziehen.

Trotz der einfachen Verhaltensregeln für die Organismen (siehe Anhang A.2, S. 36), gestaltete sich die Programmierung dieser in Java als relativ umständlich. Den meisten Aufwand bereitete in diesem Zusammenhang die Schaffung einer geeigneten Abstraktionsebene, die es ermöglichte jenseits der unteren Grafikoperationen wie Vector– und Winkelberechnungen auf der eigentlichen Bedeutungsebene von Aussagen wie z. B. „Wenn eine Mutter in der Nähe ist, dann verfolge sie in einem Abstand von x“ zu arbeiten.[1] Dieses Vorhaben ist auf Grund des knappen zeitlichen Rahmens nur teilweise gelungen, reichte allerdings für die gesteckten Projektanforderungen aus.

Da das Indoor–Positioning–System nicht eingesetzt werden konnte (siehe Kapitel 3, S. 15), fand leider keine Interaktion mit dem Besucher statt. Alternativ wurde die Simulation mit künstlichen Positionsdaten versorgt.

Für eine erneute Installation von WeFollowYou! würde auf ein Indoor–Positioning–System mit Ubisense (Ubisense, 2009, siehe Kapitel 3) verzichtet werden, da der technische Aufwand gegenüber den gewonnenen Interaktionsmöglichkeiten relativ hoch ist. Stattdessen könnte die Positionserkennung mittels Kamera und einem sensitiven Boden in Betracht gezogen werden. Zum einen, würde das System unabhängiger und eigenständiger werden. Zum anderen, könnten über die Kamera Gesten der Besucher erkannt werden. Mittels der Bodentechnologie wäre es möglich zu registrieren, wenn Besucher auf Organismen treten. Das erlaube umfangreichere Interaktionsmöglichkeiten zwischen Besucher und Organismen.

[1] Das Thema Sprachen für die Beschreibung von Agentenverhalten wird z. B. in (Tennstedt, 2009) behandelt.

3. Indoor–Positioning–System

(Sven Tennstedt)

Die Projekte *Sleeping Cocoons*, *Untitled (animal insector minor)*, *WeFollowYou!* und *magic sound tunnel* sollten mit Hilfe eines Indoor–Positioning–System erkennen, ob und wo sich Besucher relativ zu ihnen befinden.[1] Bei *Untitled (animal insector minor)* war ursprünglich geplant, dass der Roboter sogar seine eigene absolute Position übermittelt bekommt. Technische Unwägbarkeiten bei der Übertragung von Daten zum Lego NXT führten jedoch dazu, dass dieses Projekt nicht mit Positionsdaten versorgt wurde.

Im Folgenden wird das System zur *Echtzeit–Ortung* Ubisense, das im Rahmen des Projektes eingesetzt werden sollte, vorgestellt, sowie die Gesamtinstallation des Indoor–Positioning–Systems erläutert. Anschließend wird der Aufbau der Software beschrieben, die dafür zuständig war, die Positionsdaten aus dem globalen Koordinatensystem in die lokalen Koordinatensysteme der einzelnen Projekte zu transformieren und diese anschließend an die Projekte zu versenden.

3.1. Systemaufbau und Softwarelösung

Für das Projekt *Ambient Awareness* sollte das industrielle System zur *Echtzeit–Ortung* von Ubisense (Ubisense, 2009) zum Einsatz kommen. Dieses System kann mit 3 bis 8 Funksensoren arbeiten, die so positioniert werden müssen, dass sie den zu beobachtenden Bereich umschließen. Jedes Gerät, jeder Gegenstand oder jeder Mensch, dessen Position ermittelt werden soll, muss einen kleinen Funksender, den so genannten Tag, an sich tragen. Dieser Tag sendet periodisch ein Signal mit seiner Kennung aus, das im Idealfall von allen Sensoren empfangen wird. Über die unterschiedlichen Laufzeiten des Signals zu den einzelnen Sensoren kann die Position eines Tags trianguliert werden. Der Hersteller verspricht eine Genauigkeit von bis zu 15 Zentimeter.

[1] Projektbeschreibungen unter (ambientawareness, 2009)

Für die drei Projekte *Sleeping Cocoons*, *WeFollowYou!* und *magic sound tunnel* war es unpraktisch mit globalen Positionsdaten zu arbeiten. Entweder, weil sie ein lokales Koordinatensystem benötigten (*WeFollowYou!* und *magic sound tunnel*) oder weil nur interessant war, ob sich überhaupt jemand in der Nähe aufhielt (*Sleeping Cocoons*). Aus diesem Grund wurde eine Software (*LocationController*) erstellt, die diese Informationen liefern konnte. Der Aufbau dieser Software wird in Unterabschnitt 3.1.2 erläutert. Vorher wird auf den Aufbau der gesamten Installation des Indoor–Positioning–System eingegangen werden.

3.1.1. Systemaufbau

Das gesamte Indoor–Positioning–System war, wie in Abbildung 3.1 zu sehen, aufgebaut. Die Ubisense–Anlage sendete alle 500 Millisekunden die aktuellen Positionen aller erkannten Tags als *iROS–Event* über den *iROS–EventHeap*.[2] Eine andere Software konnte sich an dem *iROS–EventHeap* als Empfänger für bestimmte Eventtypen anmelden, dieser leitete dann alle eingehenden Events diesen Typs an die jeweilige Software weiter. Dies ist das Standardvorgehen, wie es im iFlat der HAW Hamburg bis dato Anwendung findet. Nach diesem Schema meldete sich der *LocationController* für den Eventtyp *UbisensePositionSystem* an und empfing dadurch alle 500 Millisekunden die aktuellen Positionsdaten. Nach der Auswertung und Transformation der Positionsdaten im *LocationController*, sendete dieser die transformierten Daten an die jeweiligen Projekte weiter.

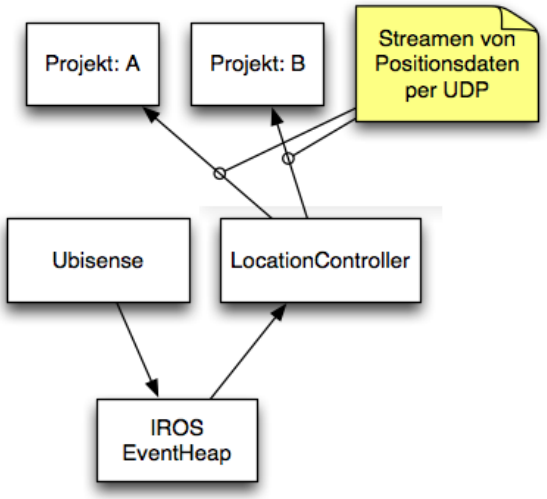


Abbildung 3.1.: Systemaufbau: Übermittlung von Positionsdaten

3.1.2. Software

Die Software besteht aus zwei Komponenten, einmal der für das Empfangen und Umrechnen der Positionsdaten und dann aus der, die die Daten über das Netzwerk verschickt. Dieser

[2] Aufbau von iROS und dessen Verwendung im iFlat der HAW Hamburg in „Konzepte für Interaktive Räume“ (Hollatz, 2007, S. 12ff) und in „Managing Information – Personal Information Environments auf Basis von iROS“ (Hollatz, 2008, S. 9ff)

Teil beschäftigt sich mit der ersten Komponente, die maßgeblich durch Tennstedt verwirklicht wurde, die zweite Komponente wurde durch Burka entwickelt (siehe (Burka, 2009)).

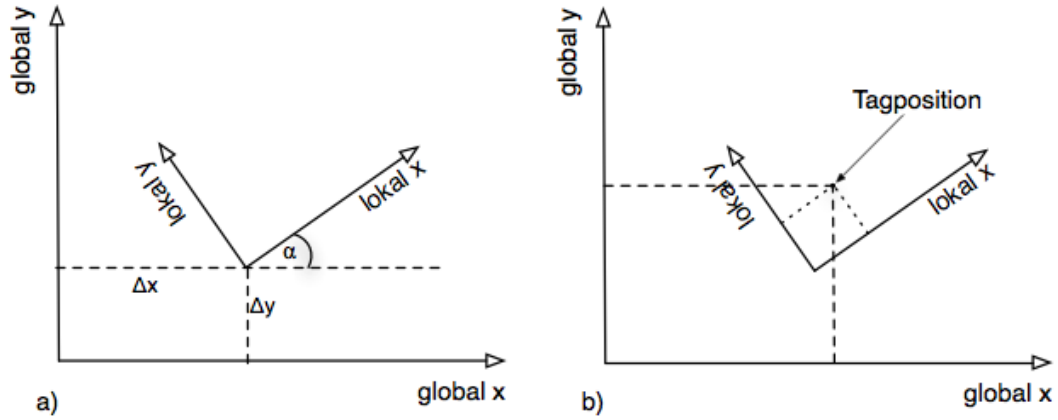


Abbildung 3.2.: a) Position und Ausrichtung des lokalen Koordinatensystems,
b) Tagposition im globalen und lokalen Koordinatensystem

Wie eingangs beschrieben wurde, gab es zwei Szenarien in den Projekten, zum einen die Anforderung Tags zu sehen, die sich in der Nähe eines Projektes aufhielten. Zum anderen zusätzlich die Umrechnung der Tagpositionen in das jeweilige lokale Koordinatensystem (siehe Abbildung 3.2, S. 17 und im Anhang B.2, S. 43 Codeauszüge).

Daraus ergaben sich für die zu entwickelnde Software folgende Konfigurationsebenen:

1. Bereich festlegen, in dem sich ein Projekt befindet und
2. Festlegen des lokalen Koordinatenursprungs und dessen Winkel im Bezug zum globalen Koordinatensystem.

Der Workflow innerhalb der Software sah folgendermaßen aus:

1. Positionsdaten empfangen,
2. überprüfen, ob sich Tags in den konfigurierten Bereichen befinden,
3. wenn ja, dann Umrechnung in das jeweilige lokale Koordinatensystem und
4. Versenden der jeweiligen lokalen Positionsdaten an die Projekte.

Abbildung 3.3 zeigt die grafische Repräsentation der Konfiguration wie sie für die Ausstellung erstellt wurde. Drei Projekte wurden konfiguriert, jedes repräsentiert durch die IP–Adresse des jeweiligen Projektrechners. Oben rechts ist der Tunnel zu sehen, der sich mit dem Bereich für WeFollowYou! direkt darunter überschneidet. Die grünen rechtwinkligen Linien zeigen das jeweilige lokale Koordinatensystem an.

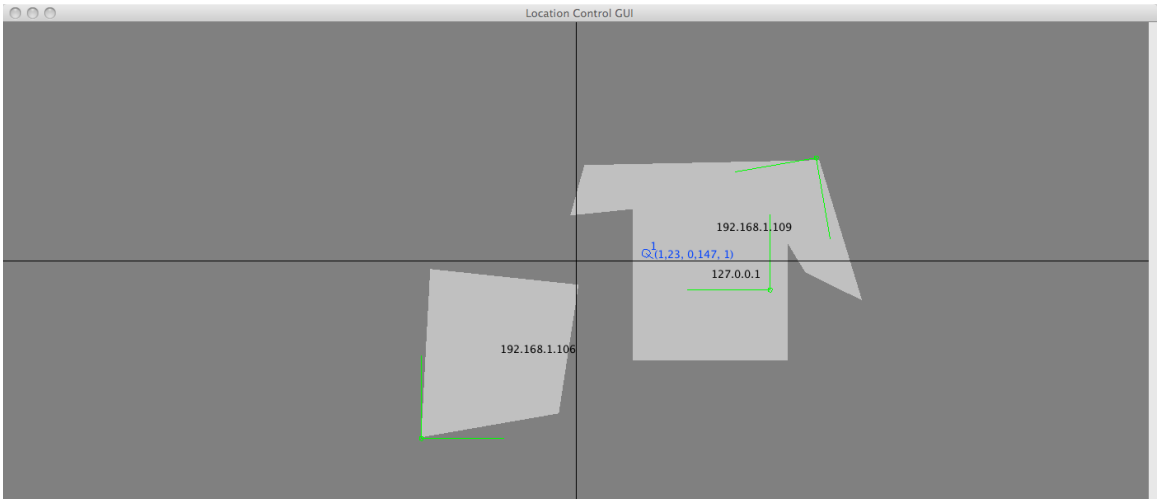


Abbildung 3.3.: Darstellung der Konfiguration von LocationController

3.2. Einsatz des Systems

Der Aufbau der Ubisense–Anlage wurde von Wendt, der kein Projektmitglied war, aber die Anlage im iFlat betreut, übernommen. Das System der HAW Hamburg bestand aus 4 Empfängern auf 4 Meter großen Stativen, die in den äußersten Ecken der Aula des Departments *Design, Medien und Information* aufgestellt wurden. Damit das System genaue Werte ermitteln konnte, mussten die Abstände der Empfänger sorgfältig ermittelt werden. Dies geschah mittels Maßbändern, für die höchste Genauigkeit empfiehlt der Hersteller jedoch eine Vermassung mit Lasermessgeräten. Anschließend wurde die Anlage mit Hilfe eines Tags, der sich dafür im Zentrum der vier Empfänger befinden musste, geeicht.

Bisher gab es keine Erfahrung mit einer Installation dieser Größenordnung. Daher war die erzielte Genauigkeit nach der ersten groben Konfiguration mit ca. 0.5 Metern unerwartet hoch.

Mit dieser Konfiguration der Ubisense–Anlage wurde die Funktion von *LocationController* getestet. Die Ermittlung der einzelnen Bereiche geschah mittels eines Tags, das ein Helfer zu den jeweiligen Eckpositionen eines Projektes trug. Die Koordinaten wurden über die *LocationControllerGUI* ermittelt (Abbildung 3.3 zeigt in blau ein Tag mit der ID 1). Erste Tests mit *WeFollowYou!* und dem *magic sound tunnel* zeigten, dass die Software die gestellten Anforderungen erfüllte.

Die Ubisense–Analge zeigte allerdings schon in dieser Phase Instabilitäten bei der Positionsbestimmung. Tags in Ruhelage wurden in unregelmäßigen Abständen an Positionen gemeldet, die von den tatsächlichen um bis zu 5 Metern abwichen. Danach brauchte die Anlage einige Sekunden, bis das Tag wieder an der realen Position gemeldet wurde.

Eine Feinjustierung der Anlage brachte jedoch zu Tage, dass dies an Störungen innerhalb der Anlage lag, die in der iFlat–Installation bisher nicht aufgefallen waren. Eine kurzfristig umsetzbare zufiredenstellende Lösung dieses Problems konnte bis zum Beginn der Ausstellung nicht realisiert werden. Da die Stromversorgung der Empfänger wurde über Power–over–Ethernet gewährleistet wurde, wurde letztendlich vermutet, dass der Ethernet–Switch keine ausreichende Leistung liefern konnte. Weil das Verhalten der Anlage nicht vorhersagbar und die Reaktion nach den Störungen zu träge war, wurde entschieden, das gesamte Indoor–Positioning–System für die Ausstellung komplett abzuschalten.

4. Skinsight

(Julia Pressburger)

> „ *von Janina Mertz (Produktdesign Textil), Julia Pressburger (Informatik/Quartz Composer) & Henryk Wollik (Kommunikationsdesign/Adobe Flash)*
>
> Zur Gestaltung einer pulsierenden Oberfläche des Organismus wählten wir für das SKINSIGHT Projekt die Methode des Motion Capturing. Eine reglose Fläche kann aktiviert werden, indem menschliche Bewegungen darauf projiziert werden. Dabei löst der Betrachter das Erscheinen von kleinen Partikeln aus, die mittels Adobe Flash und dem Quartz Composer in Form von “hautschützenden” Gebilden bewegbar und erlebbar gemacht werden. “ (ambientawareness, 2009)

Dieses Kapitel beschäftigt sich mit der Installation *Skinsight*. Das Zitat oben stammt aus dem Ausstellungsführer und kann auch auf der Webseite zu Ambient Awareness nachgelesen werden. Verfasst wurde die Beschreibung von Janina Mertz. Sie beschreibt gut die Idee die hinter der Installation steht.

Der Organismus soll auf die Besucher reagieren. Die Idee für Janinas Installation war, dass auf einem Teil des *„emotional Tent“* die Haut auf Besucher reagieren sollte, ein Pulsieren oder Atmen sollte immer sichtbar sein. Wenn nun ein Besucher in die Nähe der Haut kommt, sollte diese mit „Botenstoffe“ oder „Nervenimpulse“ reagieren, .

Für die Realisierung wählten wir einen Aufbau mit Webcam und Beamer. Die Webcam sollte einen Bereich vor der Installation aufnehmen. Wenn ein Besucher diesen Bereich betritt oder durchschreitet sollte diese Bewegung erkannt werden.

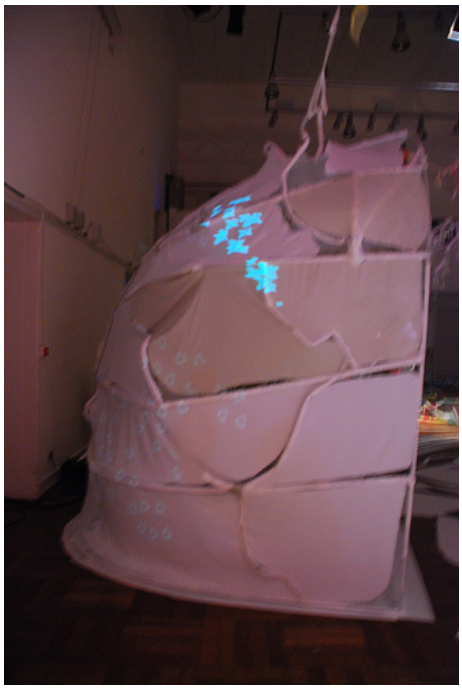


Abbildung 4.1.: **Skinsight** AmbientAwareness.org

(a) Testen (b) Aufbau (c) Eröffnung

Abbildung 4.2.: Fotos von Skinsght ©Julia Pressburger

Der Beamer sollte auf die Außenseite des Zeltes strahlen. Wenn keine Bewegung von der Kamera festzustellen ist, sollte er die Zellatmung auf die Verkleidung projizieren. Bei Bewegung sollten animierte Partikel hinzukommen. Die Partikel sollten nicht irgendwo im Bild auftauchen. Wo sie auf dem Zelt auftauchen würden sollte damit zusammenhängen, wo im von der Kamera überwachten Bereich, eine Bewegung festgestellt worden war.

In Kapitel 6, Bewegungserkennung, wird auf die Technik und die Realisierung des Projektes näher eingegangen.

5. Little Vintage Garden

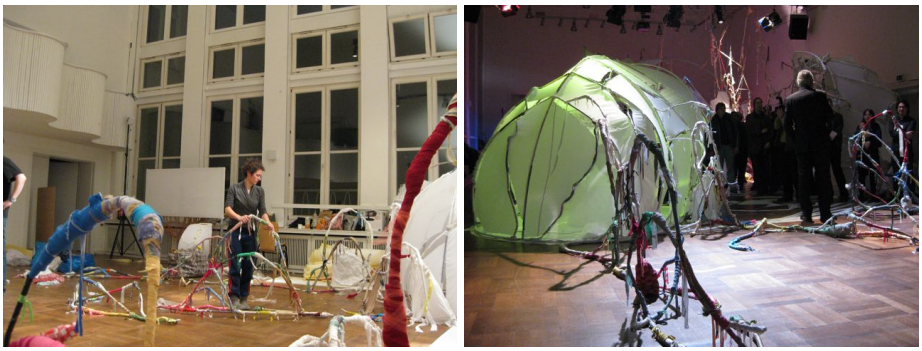
(Julia Pressburger)

> „ *Mitwirkende: Julia Pressburger (Informatik), Judith Stryczek (Kostümdesign), Henryk Wollik (Kommunikationsdesign), Oliver Dreschke (Informatik), Sebastian Gregor (Informatik)*
>
> In dieser Landschaft wird die Bewegung des Betrachters hörbar gemacht. Eine Kamera fängt die Bewegung ein und löst damit Soundschnipsel aus, wobei die Audiosignale aufgrund von verschiedenen Bewegungsabläufen unterschiedlich klingen. Parallel entsteht mittels Servomotoren und Abstandssensoren eine Eigenbewegung.
>
> Verwendete Software/Technologien: MacMini, einfache Webcam, Quartz Composer, OSCulator, Live, Arduinos, kapazitive Abstandssensoren, Servos “ (ambientawareness, 2009)

Das Zitat oben stammt aus dem Ausstellungsführer und kann auch auf der Webseite zu Ambient Awareness nachgelesen werden. Die Beschreibung der Installation stammt von Judith Stryczek. Die Idee für die Installation war, dass der Besucher durch den Garten geht, und dabei unterschiedliche Reaktionen auslöst. Mal sollte es Bewegung sein, mal Töne und Klänge und manchmal sollten Bewegung und Töne gemeinsam die Reaktion auf die wandelnden Besucher sein. Für die Ausstellung haben wir es geschafft die Bewegung des Gartens und das Spielen von Sound einzeln zu realisieren.

Abbildung 5.1.: **Little Vintage Garden** AmbientAwareness.org

Der Sound im *Little Vintage Garden* wird durch die Bewegung der Besucher ausgelöst. Wie auch bei*Skinsight* ist eine USB-Kamera auf den Garten gerichtet und zeichnet ihn auf. Das Kamerabild wird nach Bewegung durchsucht, wenn es Bewegung gibt, wird



(a) Aufbau (b) Eröffnung

Abbildung 5.2.: Fotos von Little Vintage Garden ©Julia Pressburger

ein Sound ausgelöst. Je nachdem in welchem Teil des Gartens die Bewegung war, spielt ein anderer Sound.

In Kapitel 6, Bewegungserkennung, wird auf die Technik und die Realisierung des Projektes näher eingegangen.

6. Technologie und Realisierung für die Bewegungserkennung durch eine USB-Kamera

(Julia Pressburger)

Wie schon in der Einführung kurz angeschnitten, wurde bei zwei Einzelprojekten der Ausstellung entschieden die Besucherinteraktion mit Bewegungserkennung zu gestalten. Für die Bewegungserkennung sollte eine einfache USB-Kamera verwendet werden. Die verwendete Technologie musste dazu in der Lage sein, das Kamerabild in Echtzeit zu analysieren und auf gefundene Bewegung zu reagieren.

6.1. Technologie

In diesem Kapitel gibt es eine Einführung in die verwendeten Technologien. Für weiterreichende Informationen siehe auch (Pressburger, 2009b,a; Sukale, 2008).

6.1.1. Software

Quartz Composer

Für die Realisierung der Bewegungserkennung fiel die Wahl auf den Quartz Composer, eine grafische Entwicklungsumgebung aus dem Hause Apple. Sie liegt den Xcode Tools bei, dem Development-Softwarepaket, das Apple mit seinem Betriebsystem ausliefert oder was nach kostenloser Registrierung auf der Seite der ADC (Apple Developer Connection) herunterzuladen ist.

Open Sound Control (OSC)

Für die Interprozesskommunikation wurde Open Sound Control (OSC) gewählt. Es sollte einfach zu implementieren und möglicher Container für MIDI-Daten sein. Auch im Quartz Composer ist OSC implementiert.

6.1.2. Hardware

Die Hardware-Wahl für die beiden Installationen war auf Grund der Entwicklungsumgebung eingeschränkt.

Als Computer kam nur ein Apple-Computer in Frage. Die weiteren Hardware-Komponeneten wie:

- USB-Kamera
- externe Soundkarte für 5.1 Sound
- Beamer

mussten mit dem Betriebssystem, MacOSX 10.5, betrieben werden können.

6.2. Realisierung

Nach der Recherche und dem Ausprobieren der gewählten Technologien mussten diese für die Installationen angepasst werden. Bei den Anpassungen traten kleine Probleme auf, diese und ihre Lösungen werden im Kapitel beschrieben. Im Kapitel 6.3, Architektur wird kurz beschrieben wie die eigentliche Bewegungserkennung realisiert wurde.

6.2.1. Software

Quartz Composer

Erstellte Programme im Quartz Composer heißen Compositions. Compositions bestehen aus Patches. Patches sind wie Sub-Rontinen in textuellen Programmiersprachen wie z. B. Java. Sie haben Eingänge und Ausgänge und vor der Ausgabe ist Manipulation und/oder Kombination der Eingangsdaten möglich.

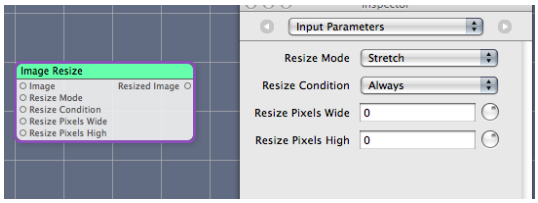


Abbildung 6.1.: **Image Resize Patch** Quartz Composer

Little Vintage Garden Die Composition für *Little Vintage Garden* ist eine kleine Abwandlung des Webcam-Pianos von dem Medienkünstler Memo. Er hatte für eine große interaktive Installation (PI Glastonbury 2008, für weitere Infos siehe auch (Akten, 2008; Grant, 2008)) das Webcam-Piano im Quartz Composer realisiert. Für *Little Vintage Garden* Garden wurde das Piano etwas vereinfacht. Es war keine visuelle Representation nötig. So wurde die von der Kamera einzusehenden Bereiche in Quadrate eingeteilt. Den Quadraten, in dehnen Judith Stryczek Soundreaktion wollte, wurden MIDI-Noten zugewiesen.

Skinsight Am Beispiel des Webcam-Pianos konnte man eine einfache Implementierung von Bewegungserkennung mit dem Quartz Composer sehen. An diesem Beispiel orientiert wurde die Composition für *Skinsight* aufgebaut. Für die Architektur der Bewegungserkennung siehe Kapitel 6.3 und (Pressburger, 2009b).

Zusätzliche Software für Little Vintage Garden

Die Realisierung von *Little Vintage Garden* brauchte noch zusätzlich Software. Die Installation reagiert, anders als *Skinsight*, mit Sound. Die benötigte Software und Konfiguration wird im folgenden Abschnitt kurz beschrieben.

OSCulator OSCulator ist eine Software, die OSC-Nachrichten empfangen kann und diese, z. B. als MIDI, in einem anderen Format weitersendet. Für das Projekt brauchten wir die Funktion MIDI-Noten, die in OSC-Pakete gepackt waren, auszulesen und diese dann auf dem MIDI-Kanal auszugeben.

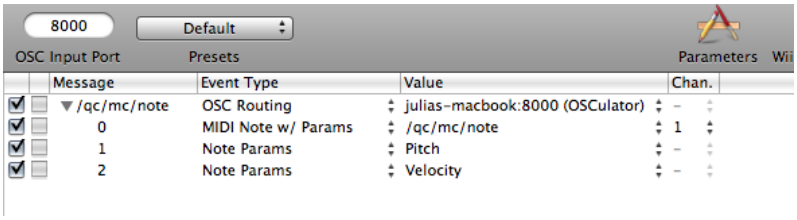


Abbildung 6.2.: **OSCulator** MIDI-Note mit Parametern

Der OSCulator ist keine freie Software, doch da die OSC-Implementierung im Quartz Composer nur eingeschränkt ist (siehe (Pressburger, 2009b), Kapitel 2.2), war dies die schnellste Möglichkeit MIDI-Daten zu bekommen.

Audio-MIDI-Setup Beim Testen der Bewegungserkennung für die Sound-Reaktion kam zuerst kein Sound. Die Recherchen ergaben dass bei MacOSX für das Nutzen von MIDI-Instrumenten noch Einstellungen im Audio-MIDI-Setup notwendig sind.

Das Audio-MIDI-Setup ist ein Teil des Betriebssystems auf dem die Installation zu *„Little Vintage Garden“* lief. Um MIDI-Kommunikation zwischen unterschiedlichen Programmen auf der Machine zu ermöglichen muss der IAC-Treiber (IAC –> Interapplication Communications) aktiviert sein.

Im Audio-MIDI-Setup wäre es auch möglich die MIDI-Nachrichten nicht lokal auf einen Kommunikationsbus zu legen, sondern per Netzwerktreiber an einen entfernten Rechner zu schicken, der dann das Umsetzten der MIDI-Nachricht übernehmen könnte.

Live Live ist eine Software die im Projekt *„Little Vintage Garden“* als MIDI-Prozessor fungierte.

> *„ ... Live in der Lage, MIDI-Geräte in die Performance einzubinden. Somit erlaubt Live das Komponieren / Improvisieren von Musik in Echtzeit, wobei die Software auch über so genannte MIDI-Controller (z. B. Tastatur, MIDI-Keyboard, Mischpult etc.) gesteuert werden kann. Die Fernsteuerung über MIDI erlaubt Musikern verschiedene Manipulationen zur selben Zeit... “* (wiki-09c)

Für die Installation wurde die Möglichkeit zur Fernsteuerung über MIDI genutzt.

Die MIDI-Noten aus dem Webcam-Piano wurden mit unterschiedlichen Samples belegt. Die Bewegung vor der Kamera stieß so, je nachdem wo die Bewegung im *Little Vintage Garden* war, unterschiedliche Sound-Samples als Reaktion an.

6.2.2. Hardware

Am Tag vor der Ausstellung wurden die Installationen so aufgestellt wie sie für die Dauer der Ausstellung bleiben sollten. Für die USB-Kameras wurden dabei auch 5m USB-Verlängerungskabel verwendet. Mit Verlängerungen waren die Installationen vorher nicht getestet worden. Jetzt gab es Probleme mit Latenzzeiten bei der Übertragung des Kamerabildes. Für *Skinsight* konnte die Kamera weiter untern an der Installation angebracht werden und das Problem der Latenz aufgehoben werden. Für *Little Vintage Garden* ging das nicht, dass führte bei der Reaktion des Systems zu einem leicht emergenten Verhalten.

Skinsight Um performant zu laufen brauchte *Skinsight* einen MacPro. Die Grafikkarte im MacPro war mächtiger und schafte es ohne ruckeln die Projection des Particle-Systems zu rendern. Das Macbook auf dem die Composition entwickelt wurde war gegen Ende der Entwicklungsphase mit dieser Aufgabe überfordert.

Little Vintage Garden Für *Little Vintage Garden* reichte ein MacMini aus.

6.3. Architektur

Mit dem Quartz Composer konnte die Bewegungserkennung überraschend einfach realisiert werden. Ohne große Vorkenntnisse über die Betriebssystem-Architektur und die Implementierung und Eigenschaften der Grafikbibliotheken, konnten vorhandene Patches zu einer

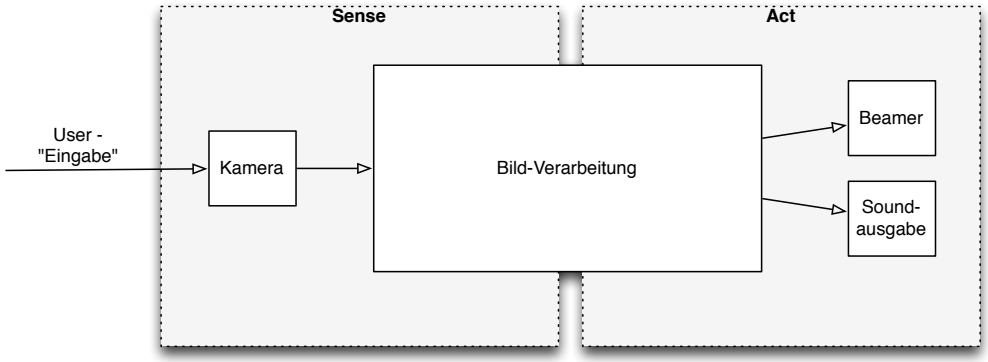


Abbildung 6.3.: **Sense-Act-Modell**

Composition zusammengestellt werden. Die Compositions können das Kamerabild analysieren und Positionsdaten per OSC-Sender ausgeben oder für *Little Vintage Garden* aus den Positionsdaten MIDI-Noten erstellte und diese dem OSC-Sender übergeben. Abbildung 6.3 stellt das reaktive System schematisch dar. Das Kamerabild durchläuft einige Stufen damit daraus die Bewegung erkannt werden kann.

Es wird zuerst in ein Graustufen-Bild umgewandelt. Das entstandene Bild wird vom vorher gemachten Bild abgezogen, ein Differenzbild erstellt. Es entsteht ein Bild das vorrangig schwarz ist, nur die Unterschiede der beiden Bilder erscheinen in unterschiedlichen Grautönen.

Dieses Bild wird jetzt verkleinert, denn der folgende Arbeitsschritt ist sehr rechenintensiv. Je detaillierter Bewegung erkannt werden soll, umso grösser muss das Bild für den nächsten Schritt bleiben.

Das verkleinerte Bild wird jetzt Pixel für Pixel durchgegangen. Der RGB-Wert des Pixels wird mit einem Schwellenwert verglichen der vorher festgelegt wurde. Ist der Wert überschritten wird der RGB-Wert und die Position des Pixels im Bild weitergereicht. Diese drei Werte bilden die Basis für die Positionsdaten des Particle-Systems, an welcher Stelle die Partikel auf der Projektion erscheinen sollen und welche MIDI-Note Live empfängt. (Pressburger, 2009b)

6.4. Fazit

Die Wahl der Bewegungserkennung als Sensor für *Skinsight* und *Little Vintage Garden* zu nutzen war richtig. Eine prototypische Lösung war mit den gewählten Komponenten und Technologien, trotz extremen Zeitdrucks, zu erreichen. Mit mehr Zeit bis zur Ausstellungen

wäre die Abstimmung zwischen Aktion der Besucher und Reaktion des Systems feiner und das Interaktionserlebnis intensiver gewesen.

7. Schluss

An dem Projekt waren insgesamt 27 Personen beteiligt, davon 23 Studenten. Die Studenten kamen im Verhältnis von ca. ein Drittel zu zwei Dritteln aus den Departments *Design, Medien und Information* und *Technik und Informatik*. In und um drei Zelte, die ebenfalls erst entworfen werden mussten, wurden acht Einzelprojekte auf die Beine gestellt.

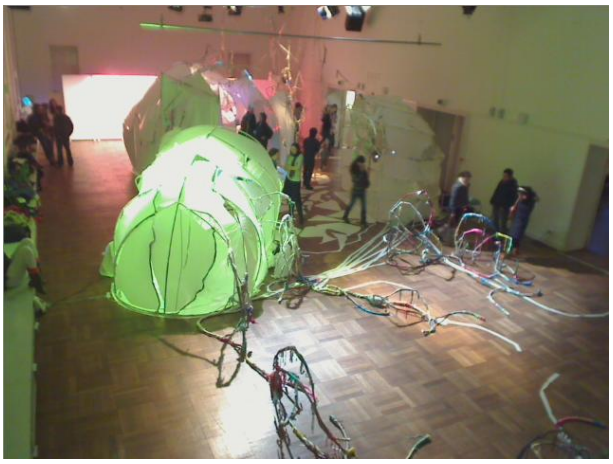


Abbildung 7.1.: **Ausstellungsaubau** (ambientawareness, 2009)

Für ein Projekt dieser Dimension ist ein Semester eigentlich viel zu knapp angesetzt und kaum zu realisieren. Doch gemessen an der Ausstellung und der Resonanz der Besucher war das Projekt ein voller Erfolg. Das ist sicherlich vor allem der hohen Motivation der Projektteilnehmer zu verdanken, die weit über die im Studienführer angesetzten Stunden an dem Projekt gearbeitet haben. Dies wiederum wurde durch die tatkräftige Unterstützung der betreuenden Professoren getragen.

Interessant war, dass den Besuchern der Ausstellung die Fehler, die die Techniker vor der Eröffnung als gravierend eingestuft haben, gar nicht erst aufgefallen waren. Als Beispiel ist das kurzfristige Wegfallen des Indoor–Positioning–Systems anzuführen. Dieses war essenzieller Bestandteil für die Interaktion in mehreren Projekten, wie u. a. WeFollowYou!. Da die Besucher aber keine Kenntnis davon hatten, ist es ihnen nicht negativ aufgefallen und sie waren von der Wirkung der Lebendigkeit der Organismen begeistert.

Im Laufe des Projektes fiel auf, dass Informatiker und Designer aus verschiedener Art Holz geschnitzt sind und auf unterschiedliche Weise an ihre Arbeit heran gehen. Während die Informatiker es durch den Aufbau ihres Studiums gewohnt sind im Team zu arbeiten, steht im Studium der Designer scheinbar der eigene Stil und die eigene und damit alleinige Arbeit im Vordergrund.

Für zukünftige Projekt wäre es gut, wenn jemand auf das Zusammenspiel der Einzelprojekte im Gesamtkonzept achten würde. Es fehlte an Projektleitung und Führung. Auch hing es stark von den Kompetenzen der einzelnen Teammitglieder ab, wie sich die individuellen Arbeiten an den Einzelprojekten später zusammenfügten.

Literaturverzeichnis

[wiki-09c] *Ableton Live*. – URL http://de.wikipedia.org/wiki/Ableton_Live. – Zugriffsdatum: 7. März 2009

[ambientawareness 2009] *AmbientAwareness - Projects*. – URL http://ambientawareness.org/projects/. – Zugriffsdatum: 3.März 2009

[Amptown 2009] *Amptown*. – URL http://www.amptown.de/. – Zugriffsdatum: 02.03.2009

[Ubisense 2009] *Ubisense: Echtzeit–Ortung*. – URL http://www.ubisense.de/. – Zugriffsdatum: 05.03.2009

[Adamczyk u. a. 2007] ADAMCZYK, Piotr D. ; HAMILTON, Kevin ; TWIDALE, Michael B. ; BAILEY, Brian P.: HCI and new media arts: methodology and evaluation. In: *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2007, S. 2813–2816. – ISBN 978-1-59593-642-4

[Akten 2008] *Pi @ Glastonbury 2008*. – URL http://memo.tv/projects/pi_glastonbury_2008. – Zugriffsdatum: 27.02.2009. AKTEN, Mehmet S.

[Burka 2009] BURKA, Florian: Projekt Florian Burka Ambient Awareness - Federfarbener Tunnel / HAW Hamburg. Berliner Tor 7, 2009. – Forschungsbericht

[Grant 2008] *Pi @ Glastonbury 2008*. – URL http://flickr.com/photos/evangrant/2645073797/. – Zugriffsdatum: 27.02.2009. GRANT, Evan

[Hollatz 2007] HOLLATZ, Dennis: Konzepte für Interaktive Räume / HAW Hamburg. Berliner Tor 7, July 2007. – Forschungsbericht

[Hollatz 2008] HOLLATZ, Dennis: Managing Information – Personal Information Environments auf Basis von iROS / HAW Hamburg. Berliner Tor 7, March 2008. – Forschungsbericht

[Pressburger 2009a] PRESSBURGER, Julia: Digital Art Design / HAW Hamburg, Department Informatik. URL http://users.informatik.haw-hamburg.de/~ubicomp, February 2009. – Forschungsbericht

[Pressburger 2009b] PRESSBURGER, Julia: Interaktive Kunst / HAW Hamburg, Department Informatik. URL http://users.informatik.haw-hamburg.de/~ubicomp, February 2009. – Forschungsbericht

[Reas und Fry] *Processing.org*. – URL http://processing.org/. – Zugriffsdatum: 27.02.2009. REAS, Casey ; FRY, Ben

[Sukale 2008] SUKALE, Martin: *Konstruktion eines Netzwerkes eingebetteter Systeme für interaktives Design*. www.informatik.haw-hamburg.de, HAW Hamburg, Department Informatik, Diplomarbeit, August 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/sukale.pdf

[Tennstedt 2009] TENNSTEDT, Sven: Domain Specific Languages für die Beschreibung von Domain Specific Languages für die Beschreibung von Agentenverhalten / HAW Hamburg. Berliner Tor 7, February 2009. – Ausarbeitung

[Zhang und Candy 2007] ZHANG, Yun ; CANDY, Linda: An in-depth case study of art-technology collaboration. In: *C&C '07: Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*. New York, NY, USA : ACM, 2007, S. 53–62. – ISBN 978-1-59593-712-4

A. WeFollowYou!

A.1. Processing Beispiel

Ein Beispiel aus Processing - **Brightness**:

```
/**
 * Brightness
 * by Rusty Robison.
 *
 * Brightness is the relative lightness or darkness of a color.
 * Move the cursor vertically over each bar to alter its brightness.
 */

int barWidth = 5;
int[] brightness;

void setup()
{
  size(200, 200);
  colorMode(HSB, 360, height, height);
  brightness = new int[width/barWidth];
}

void draw()
{
  int j = 0;
  for (int i = 0; i <= (width-barWidth); i += barWidth) {
    noStroke();
    if ((mouseX > i) && (mouseX < i+barWidth)) {
      brightness[j] = mouseY;
    }
    fill(i, height, brightness[j]);
    rect(i, 0, barWidth, height);
    j++;
  }
}
```

A.2. Verhaltensregeln der Organismen

Hier zum Vergleich die Verhaltensregeln für Mütter und für Mikroben. Während das Verhalten der Mütter mit nur vier Regeln ausreichend definiert werden konnte, mussten für die Mikroben drei Zustände mit drei bis fünf Regeln definiert werden. Die Regeln sind ihrer Ausführpriorität nach geordnet. Eine Regel, die erfüllt wird, verdrängt alle anderen Regeln, die eine niedrigere Priorität besitzen.

Mutter:

1. Erkennung des Weltrandes
2. weiche anderen Müttern aus: Min–Distanz 50 Pixel
3. weiche Hort aus: Min–Distanz 50 Pixel
4. schwimme umher

Mikrobe:

- **Wenn** *in freiem Gewässer*:
    1. Erkennung des Weltrandes
    2. weiche anderen Mikroben aus: Min–Distanz 25 Pixel
    3. **Wenn** erschöpft:
        - Schwimm zu Hort, **wenn** in Sichtweite, und wechsel in Zustand *im Hort*

        sonst:

        - Schwimm zu Menschen, **wenn** in Sichtweite, und wechsel in Zustand *beim Menschen*
    4. folge Mutter, wenn in Sichtweite
    5. schwimme umher
- **Wenn** *im Hort*:
    1. Erkennung des Weltrandes
    2. weiche anderen Mikroben aus: Min–Distanz 12 Pixel
    3. **Wenn** noch erschöpft:
        - Schwimm um Hortmittelpunkt

        sonst:

- wechsel in Zustand *in freiem Gewässer*

- **Wenn** *beim Menschen*:
  1. Erkennung des Weltrandes
  2. weiche anderen Mikroben aus: Min–Distanz 25 Pixel
  3. **Wenn** Mensch noch in Sichtweite:
     - schwimme um den Menschen

     sonst:
     - wechsel in Zustand *in freiem Gewässer*

B. Indoor–Positioning–System

B.1. LocationController Konfiguration

Die Konfiguration von *LocationController* geschah mit Hilfe des Spring–Frameworks. Die Folgende Datei stellt die Konfiguration für die AmbientAwareness–Ausstellung dar.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
␣␣␣␣␣␣␣http://www.springframework.org/schema/beans
␣␣␣␣␣␣␣http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="worldBounds" class="util.quadtree.Bounds">
    <constructor-arg index="0">
      <value type="java.lang.Float">5</value>
    </constructor-arg>
    <constructor-arg index="1">
      <value type="java.lang.Float">-10</value>
    </constructor-arg>
    <constructor-arg index="2">
      <value type="java.lang.Float">-5</value>
    </constructor-arg>
    <constructor-arg index="3">
      <value type="java.lang.Float">10</value>
    </constructor-arg>
  </bean>

  <bean id="heapLocation" class="java.lang.String">
    <constructor-arg value="10.0.2.72" />
  </bean>

  <bean id="swarm" class="positioning.geom.AreaBound">
    <constructor-arg index="0">
      <list>
        <!-- links oben -->
        <bean class="javax.vecmath.Point2f">
          <constructor-arg index="0">
            <value type="java.lang.Float">1.0</value>
          </constructor-arg>
```

```
        <constructor-arg index="1">
          <value type="java.lang.Float">-2.1</value>
        </constructor-arg>
      </bean>
      <!-- rechts oben -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">1.0</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">1.4</value>
        </constructor-arg>
      </bean>
      <!-- rechts unten -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">3.7</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">1.4</value>
        </constructor-arg>
      </bean>
      <!-- links unten -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">3.7</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">-2.1</value>
        </constructor-arg>
      </bean>
    </list>
  </constructor-arg>
  <constructor-arg index="1">
    <bean class="positioning.geom.LocalCoord">
      <constructor-arg index="0">
        <value type="java.lang.Float">90</value>
      </constructor-arg>
      <constructor-arg>
        <bean class="javax.vecmath.Vector2d">
          <constructor-arg index="0">
            <value type="java.lang.Double">2.4</value>
          </constructor-arg>
          <constructor-arg index="1">
            <value type="java.lang.Double">1.5</value>
          </constructor-arg>
        </bean>
      </constructor-arg>
```

```
    </bean>
  </constructor-arg>
</bean>

<bean id="tunnel" class="positioning.geom.AreaBound">
  <constructor-arg index="0">
    <list>
      <!-- Eingang links -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">0.15</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">2.0</value>
        </constructor-arg>
      </bean>
      <!-- curve aussen -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">4.25</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">2.1</value>
        </constructor-arg>
      </bean>
      <!-- Aussgang rechts aussen -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">4.99</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">-0.84</value>
        </constructor-arg>
      </bean>
      <!-- Aussgang links innen -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">4.00</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">-0.25</value>
        </constructor-arg>
      </bean>
      <!-- curve innen -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">3.2</value>
        </constructor-arg>
```

```
        <constructor-arg index="1">
          <value type="java.lang.Float">1.35</value>
        </constructor-arg>
      </bean>
      <!-- Eingang rechts-->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">-0.09</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">0.95</value>
        </constructor-arg>
      </bean>
    </list>
  </constructor-arg>
  <constructor-arg index="1">
    <bean class="positioning.geom.LocalCoord">
      <constructor-arg index="0">
        <value type="java.lang.Float">190</value>
      </constructor-arg>
      <constructor-arg>
        <bean class="javax.vecmath.Vector2d">
          <constructor-arg index="0">
            <value type="java.lang.Double">4.30</value>
          </constructor-arg>
          <constructor-arg index="1">
            <value type="java.lang.Double">3.0</value>
          </constructor-arg>
        </bean>
      </constructor-arg>
    </bean>
  </constructor-arg>
</bean>

  <bean id="cocoon" class="positioning.geom.AreaBound">
  <constructor-arg index="0">
    <list>
      <!-- rechts vorne -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">-2.54</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">-0.17</value>
        </constructor-arg>
      </bean>
      <!-- links vorne -->
      <bean class="javax.vecmath.Point2f">
```

```
        <constructor-arg index="0">
          <value type="java.lang.Float">0.046</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">-0.5</value>
        </constructor-arg>
      </bean>
      <!-- hinten links -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">-0.3</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">-3.2</value>
        </constructor-arg>
      </bean>
      <!-- hinten rechts -->
      <bean class="javax.vecmath.Point2f">
        <constructor-arg index="0">
          <value type="java.lang.Float">-2.7</value>
        </constructor-arg>
        <constructor-arg index="1">
          <value type="java.lang.Float">-3.71</value>
        </constructor-arg>
      </bean>
    </list>
  </constructor-arg>
  <constructor-arg index="1">
    <bean class="positioning.geom.LocalCoord">
      <constructor-arg index="0">
        <value type="java.lang.Float">0</value>
      </constructor-arg>
      <constructor-arg>
        <bean class="javax.vecmath.Vector2d">
          <constructor-arg index="0">
            <value type="java.lang.Double">0.0</value>
          </constructor-arg>
          <constructor-arg index="1">
            <value type="java.lang.Double">0.0</value>
          </constructor-arg>
        </bean>
      </constructor-arg>
    </bean>
  </constructor-arg>
</bean>

<!-- Starting all stuff -->
<bean id="locationController" class="positioning.LocationController">
```

```
    <constructor-arg index="0">
      <ref bean="heapLocation" />
    </constructor-arg>
    <constructor-arg index="1">
      <ref bean="worldBounds" />
    </constructor-arg>
    <property name="heapLocation" ref="heapLocation"></property>
    <property name="areaList">
      <map>
        <entry>
          <key>
            <value>127.0.0.1</value>
          </key>
          <ref bean="swarm" />
        </entry>
        <entry>
          <key>
            <value>192.168.1.109</value>
          </key>
          <ref bean="tunnel" />
        </entry>
        <entry>
          <key>
            <value>192.168.1.106</value>
          </key>
          <ref bean="cocoon" />
        </entry>
      </map>
    </property>
  </bean>
</beans>
```

B.2. Transformation ins lokale Koordinatensystem (Code)

Codeausschnitt aus der Klasse, die das lokale Koordinatensystem repräsentiert:

```
\begin{lstlisting}
/**
 * @author Sven Tennstedt
 *
 */
public class LocalCoord {
  // ...
  private PolarLine yOrdi = null;
  private PolarLine xOrdi = null;
```

```
  // ...

  /**
   * Ein Punkt innerhalb einer Bounding Box wird in das
   * Ziel System transformiert.
   *
   * @param toTransform
   * @return der transformierte Punkt
   */
  public Vector2f transform(Vector2f toTransform) {
    Vector2f result = new Vector2f();

    result.y = xOrdi.pointDistance(toTransform.x, toTransform.y);
    result.x = -yOrdi.pointDistance(toTransform.x, toTransform.y);

    return result;
  }

  // ...

}
```

Code für eine Gerade die mit Polarkoordinaten repräsentiert wird:

```
// package und import

/**
 * @author Sven Tennstedt
 *
 */
public class PolarLine {

  /**
   * Der Winkel der Geraden mathematisch positiver Sinn
   * Radians
   */
  private float alpha = 0;

  private Point2f position = new Point2f(0, 0);

  public PolarLine(float _alpha) {
    this.alpha = _alpha;
  }

  public PolarLine(float _alpha, Point2f position) {
    this(_alpha);
    this.position = new Point2f(position);
  }
```

```
  public float pointDistance(float x, float y) {
    Vector2f p = new Vector2f(x, y);
    p.sub(position);
    float angle = MathUtil.angleFrom(p);
    angle = angle - alpha;
    float distance = (float)Math.sin(angle) * p.length();

    return distance;
  }
}
```