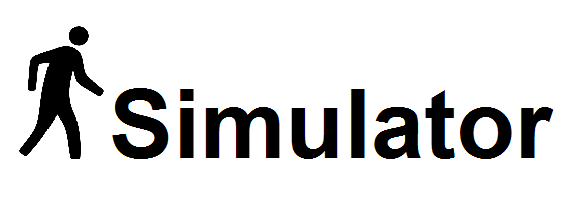
****

**Pflichtenheft**

(FuSi)

**Version 0.2**

Historie der Dokumentversionen

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Autor | Änderungsgrund / Bemerkungen |
| 0.1 | 20.09.2013 | Team | Ersterstellung |
| 0.2 | 27.10.2013 | Team | Erweiterung um Brainstorming-Experimente |
|  |  |  |  |
|  |  |  |  |

Inhaltsverzeichnis

[Historie der Dokumentversionen 2](#_Toc368057581)

[Inhaltsverzeichnis 2](#_Toc368057582)

[1 Einleitung 3](#_Toc368057583)

[1.1 Allgemeines 3](#_Toc368057584)

[1.1.1 Zweck und Ziel dieses Dokuments 3](#_Toc368057585)

[1.1.2 Ausgangslage 3](#_Toc368057586)

[2 Konzept und Rahmenbedingungen 4](#_Toc368057587)

[2.1 Ziele 4](#_Toc368057588)

[2.2 Benutzer / Zielgruppe 4](#_Toc368057589)

[2.3 Ressourcen 4](#_Toc368057590)

[2.4 Übersicht der Meilensteine 4](#_Toc368057591)

[2.5 Java Namenskonventionen 4](#_Toc368057592)

[3 Grundkonzept 5](#_Toc368057593)

[3.1 Aufbau der Simulationsumgebung (Welt) 5](#_Toc368057594)

[3.2 Ausbaustufen 5](#_Toc368057595)

[3.3 Der Landschaftseditor 6](#_Toc368057596)

[3.4 Domainmodell 6](#_Toc368057597)

[4 Detailkonzepte 7](#_Toc368057598)

[4.1 Finden der Wegpunkte eines Hindernisses 7](#_Toc368057599)

[5 Anhang / Ressourcen 8](#_Toc368057600)

# Einleitung

## Allgemeines

### Zweck und Ziel dieses Dokuments

Dieses Pflichtenheft beschreibt die Rahmenbedingungen für die geplante Fussgänger Simulation.

### Ausgangslage

Bei der Bewegung von Fussgängermengen entstehen viele interessante Phänomene von Selbst-Organistation. Das Studium dieser Phänomene ist z.B. bei der Planung von Fluchtwegen in Stadien etc. von grosser Bedeutung.

Es soll eine Software erstellt werden, welche die Bewegung von Fussgängern bei Vorhandensein von Hindernissen und Wänden simuliert. Die Bewegungen sollen grafisch visualisiert werden und es sollen wichtige Kenngrössen berechnet werden.

# Konzept und Rahmenbedingungen

## Ziele

Es soll eine Simulation von Fussgängerströmen erstellt werden. Die Fussgänger werden in einer Ihnen bekannten Landschaft positioniert. Von dieser Position aus sollen sie sich zu einem Zielpunkt bewegen. Das Verhalten bei Wegkonflikten von mehreren Fussgänger oder auch das Herdenverhalten soll damit studiert werden können.

## Benutzer / Zielgruppe

Verhaltensforscher mit erweiterten Programmierkenntnissen

## Ressourcen

Projektablage: GitHub

Dokumentation: Word / Latex

Entwicklungsumgebung: Eclipse, Idea Intellj

Java-Bibliotheken: Java 1.7, Java2D Graphics (Darstellung, Editor), XML DOM-Parser (Konfigurationsdatei)

## Übersicht der Meilensteine

|  |  |
| --- | --- |
| **Vorbereitungsphase** | |
| Installation Entwicklungsumgebung | 27.09.2013 |
| Freigabe Pflichtenheft | 27.09.2013 |
| **Implementierung und Test** | |
| Anzeigen einer Landschaft aus einer Konfigurationsdatei | 25.10.2013 |
| Ein Fussgänger marschiert konfliktfrei vom Start- zum Zielpunkt | 04.11.2013 |
| Bearbeiten von Landschaften in einem Editor | 29.11.2013 |
| Mehrere Fussgänger marschieren Ihren Weg konfliktfrei vom Start bis ins Ziel | 20.11.2013 |
| **Projektabschluss** | |
| Reserve und Abschlusspräsentation | 17.01.2013 |

## Java Namenskonventionen

* Alle Namen sind in English und in CamelCase Notation
* Variabeln haben ihren Datentyp als Präfix. Beispiele:
  + Integer: i
  + String: s
  + Objekt: o
  + Etc.
* Klassen haben ein C als Präfix und sind ein Nomen
* Interfaces haben ein I als Präfix und sind ein Nomen
* Get\* und Is\* Methoden sind immer readonly, verändern als nichts am Objekt

# Grundkonzept

## Aufbau der Simulationsumgebung (Welt)

Die Simulationsumgebung (nachfolgend bezeichnet als Welt) besteht aus einem kartesischen 2D Koordinatensystem. In dieser Welt gibt es 0 bis n Hindernisobjekte, dargestellt als Kreise, Rechtecke, Polygone, etc., sowie 1 bis n Personen, die alle einen Start- und Zielpunkt haben. Das Ziel der ganzen Simulation ist es, möglichst realitätsnah die Bewegungen der Fussgänger in der vorgegebenen Welt zu simulieren.

Zum Berechnen der Wege definieren wir folgende Grundbedingungen

* Jeder Fussgänger hat ein x-y-Ort und eine Richtung
* Jedes Hindernisobjekt stellt mit seinen äusseren Ecken einen Wegpunkt dar, anhand dem sich die Personen für Ihre Wegsuche orientieren
* Aus allen Wegpunkten generieren wir einen Graphen, bei dem keine Kante durch ein Hindernis führt.
* Mit Hilfe des Djikstra-Algorithmus errechnen wir für die Personen den kürzesten Weg durch den Graphen
* In Situationen, wo zwei Personen den gleichen Punkt betreten, wollen wir in einem ersten Schritt per Zufall entscheiden, wer gehen und wer warten muss.
* Die Simulation verfolgt das Prinzip eines Taktmodelles, in der jeder Takt z.B. eine Sekunde darstellt.

Softwaretechnisch wollen wir uns auf 2 Threads beschränken. Einer der Threads zeichnet mit Java2D das GUI, der andere führt die Simulation durch. Wir möchten mit dieser Entkoppelung erreichen, dass wir später nicht mehr zwingend jeden Schritt, sondern z.B. nur jeden dritten Schritt der Simulation zeichnen lassen können. Auf Grund einiger Tests wissen wir bereits, dass das Zeichnen mit vielen Objekten beachtliche CPU-Ressourcen in Anspruch nehmen kann und wir so eine weitere Möglichkeit haben, die Performance zu verbessern.

## Ausbaustufen

Weiter wollen wir folgende Optimierungen implementieren respektive ausprobieren:

* Über alle Wegpunkte stellen wir einen gewichteten Graph als Basisgraph zusammen. Dieser Basisgraph soll die Performance verbessern, weil so nicht jedes Mal die komplette Welt neu berechnet werden muss, sondern das Person X in den Basisgraph einsteigen kann und ihn am Punkt, der ihrem Zielpunkt am nächsten kommt, ihn wieder verlässt.
* Bei der Berechnung, welche Personen sich in der Nähe von Person x befinden, entsteht ein Algorithmus der Komplexitätsklasse O(n2). Zur Optimieren legen wir ein virtuelles Viereckraster über die Welt, in das sich jede Person in ihre jeweilige Zelle einschreibt. Dadurch können wir direkt über die eigene und 9 benachbarten Zellen auslesen, welche Personen sich in der Nähe befinden.
* Jede Personen soll über ein bestimmtes Bewegungsmodell verfügen. Damit wollen wir eine realitätsnahere Entscheidungsfindung umsetzen, als uns einfach den Zufall zu verlassen. Beispiele:
  + Der „Ellbögler“: Versucht lieber die andere Person wegzudrängen als auszuweichen
  + Der „Zurückhaltende“: Hält sich eher zurück und gibt anderen den Vortritt
  + Der „PolizistImEinsatz“: Der Dauervortritthaber
  + Der „Scheuche“: Weicht immer aus, will niemandem zu nahe kommen.
* Für jede Personen sollen statistische Daten wie zum Beispiel Verlangsamung, mittlere Geschwindigkeit, etc. festgehalten werden. Damit liesse sich dann eine Statistik erstellen, die die mittlere Geschwindigkeit oder auch der mittlere Zeitverlust in Abhängigkeit der Personen darstellt.

## Der Landschaftseditor

Mit Hilfe des Landschaftseditors soll grafisch eine Welt zusammengestellt werden können. Das Ziel ist es, dass per Drag&Drop Objekte platziert und in Form und Grösse bearbeitet werden können. Die erstelle Landschaft speichern wir in eine XML-Datei ab, die dann in der Simulation als Welt geladen werden kann.

Das XML ist durch folgendes XSD definiert:

<?xml version="1.0" encoding="utf-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name ="objList">

<xs:complexType >

<xs:sequence>

<xs:element name="obj" minOccurs="0" maxOccurs="unbounded" type="tyObj"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:complexType name="tyObj">

<xs:sequence>

<xs:element name="point" minOccurs="2" maxOccurs="unbounded" type="tyPoint"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name ="tyPoint">

<xs:attribute name ="x" use ="required" type="xs:nonNegativeInteger"/>

<xs:attribute name ="y" use ="required" type="xs:nonNegativeInteger"/>

</xs:complexType>

</xs:schema>

Beispiel XML:

<?xml version="1.0" encoding="utf-8"?>

<Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="ConfigSchema.xsd">

<objList>

<obj>

<point x="210" y="50"/>

<point x="410" y="50"/>

<point x="410" y="100"/>

<point x="220" y="100"/>

</obj>

<obj>

<point x="110" y="250"/>

<point x="210" y="200"/>

<point x="160" y="300"/>

</obj>

<obj>

<point x="310" y="170"/>

<point x="360" y="20"/>

<point x="450" y="150"/>

<point x="380" y="300"/>

<point x="330" y="280"/>

</obj>

</objList>

<walkerList>

<walker>

<source x="20" y="20"/>

<target x="520" y="300"/>

</walker>

<walker>

<source x="20" y="300"/>

<target x="520" y="20"/>

</walker>

<walker>

<source x="20" y="100"/>

<target x="520" y="300"/>

</walker>

<walker>

<source x="20" y="130"/>

<target x="520" y="300"/>

</walker>

<walker>

<source x="20" y="200"/>

<target x="520" y="300"/>

</walker>

</walkerList>

</Config>

## Domainmodell

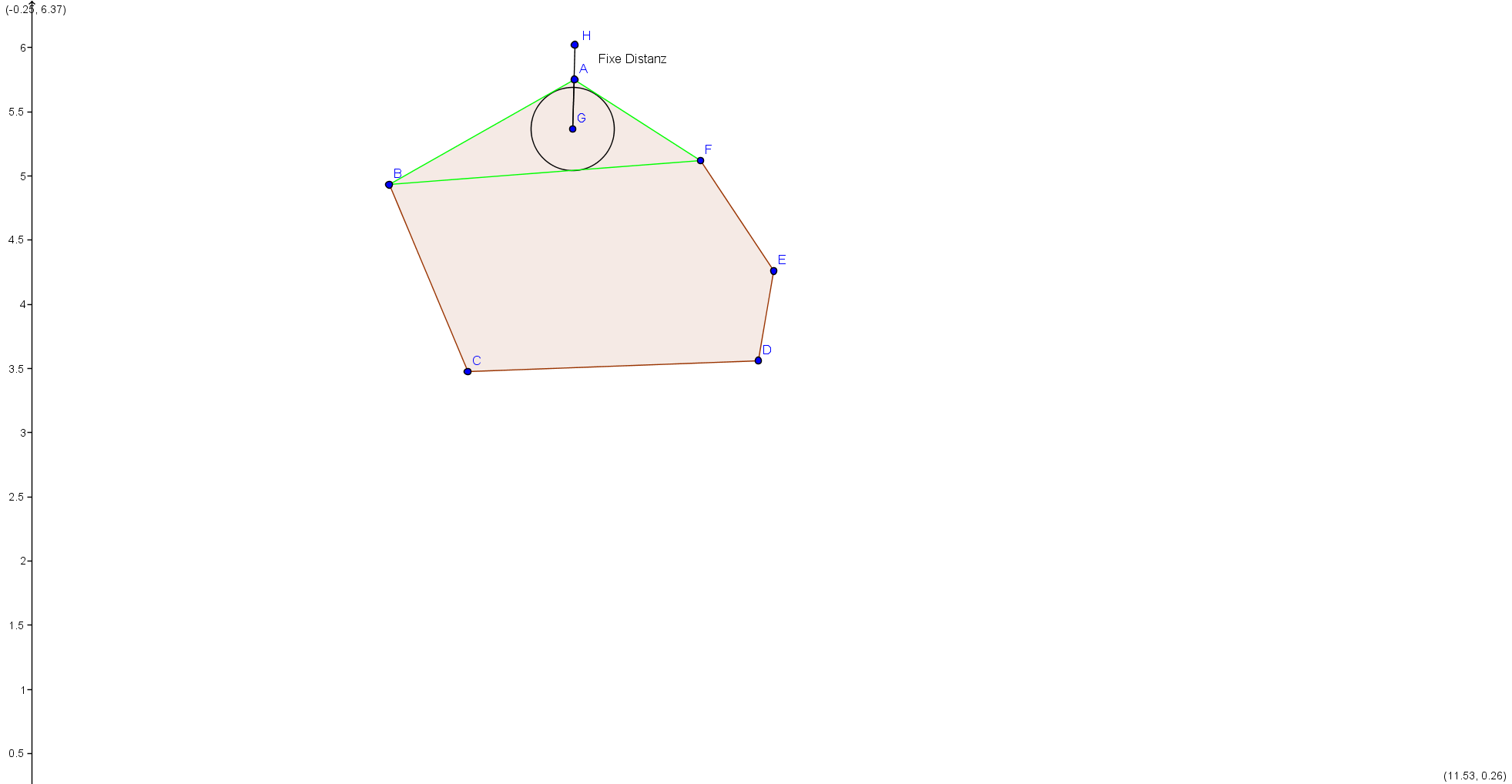
Wir bauen unsere Simulation auf folgendem Domainmodell auf:

@Bohni: Hie dis UML ifüege ^^

# Detailkonzepte

## Finden der Wegpunkte eines Hindernisses

Zum Finden eines Wegpunktes wird aus den zwei Stecken die ihm Punkten Enden ein Dreieck gebildet. Vom Inkreis-Mittelpunkt dieses Dreiecks wird zum Ecken des Hindernisses eine Strecke gezogen, diese wird dann um einen bestimmten Wert verlängert.



# Anhang / Ressourcen