

# Contents

<b>I</b>	<b>Einleitung</b>	<b>2</b>
<b>II</b>	<b>Definition: Künstliche Intelligenz</b>	<b>2</b>
<b>III</b>	<b>BlueJ</b>	<b>2</b>
<b>IV</b>	<b>Fangspiel</b>	<b>3</b>
<b>1</b>	<b>Grundaufbau</b>	<b>3</b>
1.1	Spielkonzept . . . . .	3
1.2	Klasse "Läufer" . . . . .	3
1.3	Klasse "Fänger" . . . . .	4
1.4	Anwendungsklasse "Spiel" . . . . .	4
<b>2</b>	<b>Regelbasierte Abläufe</b>	<b>5</b>
2.1	"Die Wand erkennen" . . . . .	5
2.2	"Den Läufer sichten" . . . . .	6
2.3	"Zweiten Fänger auf das Spielfeld bringen" . . . . .	7
2.4	"Die Fänger nicht aneinander geraten lassen" . . . . .	7
2.5	"Die Geschwindigkeit der Fänger erhöhen" . . . . .	9
2.6	"Der Läufer wurde gefangen" . . . . .	9
<b>V</b>	<b>Fazit</b>	<b>10</b>
<b>VI</b>	<b>Anhang</b>	<b>10</b>
<b>VII</b>	<b>Quellenverzeichnis</b>	<b>10</b>

## Part I

# Einleitung

Bereits als Kind wird man mit "künstlicher Intelligenz" konfrontiert. Erinnern sie sich noch an die ersten Roboter-Hunde, die in den Nachrichten vorgestellt wurden? Oder an das erste Computerspiel, das sie gespielt haben? Auch ich habe schon früh den Umgang mit Spielen gelernt. Seien es die Kartenspiele von Microsoft oder sogar Spiele für diverse Konsolen wie etwa der Playstation oder dem Gameboy gewesen; Mir haben diese Spiele Spaß gemacht und das machen sie auch noch heute. Aber es gibt einen entscheidenden Unterschied zwischen dem damaligen und dem heutigen Spielen dieser Spiele: Ich habe mich nicht gefragt, wie diese Spiele zustande kommen. Mittlerweile frage ich mich, wie viel Arbeit denn überhaupt dahinter stecken muss, um ein Spiel zu programmieren. Als Kind habe ich, so unbesorgt wie man eben war, einfach nur gespielt. Ich habe mir keine Gedanken darum gemacht, woher mein Gegner beim Rennspiel denn weiß, wann er bremsen muss, um die scharfe Kurve gut nehmen zu können. Noch weniger dafür, wann das Spiel denn weiß, dass ich eine Runde erfolgreich absolviert habe. Doch mittlerweile steigt mein Interesse daran, mich selbst an einem kleinen Spiel zu versuchen, um einen Einblick in den Aufwand der Programmierer zu bekommen. Deswegen möchte ich mich in dieser Facharbeit mit der "Künstlichen Intelligenz in Computerspielen" beschäftigen. Durch den Informatikunterricht an meiner Schule habe ich den Umgang mit der Entwicklungsumgebung BlueJ gelernt, mit der man auf Basis der Programmiersprache Java programmieren kann. Deswegen habe ich mir als Ziel gesetzt, diese Facharbeit mit ein über BlueJ programmiertes Spiel zu versehen und den Quelltext davon zu nutzen, um die "regelbasierten Abläufe", die ein Bestandteil der künstlichen Intelligenz in Computerspielen sind, zu erläutern.

## Part II

# Definition: Künstliche Intelligenz

Die "Künstliche Intelligenz" ist ein Forschungsgebiet, welches versucht, "menschliche Wahrnehmung und menschliches Handeln durch Maschinen nachzubilden".<sup>1</sup> Die Hauptaufgabe besteht darin, Methoden zu entwickeln, die es einem Computer ermöglichen, wie ein Mensch bestimmte Aufgaben zu lösen.<sup>2</sup> Das Forschungsgebiet steht im "ständige[n] Austausch zwischen Wissenschaftlern" der Bereiche Kognitionswissenschaft, Psychologie, Neurologie, Philosophie und Sprachwissenschaft, um aufgrund der Komplexität und den verschiedenen Anwendungsbereichen der künstlichen Intelligenz Arbeitserleichterungen zu schaffen. Die künstliche Intelligenz findet beispielsweise in den Bereichen Medizin, Bilderkennung, der Automobilindustrie oder vor allem in Spielen Verwendung. [für beide Sätze siehe 1]

---

<sup>1</sup>Kern, Sabine - <http://www.planet-wissen.de>

<sup>2</sup>Prof. Dr. Richard Lackes, Dr. Markus Siepermann - <http://wirtschaftslexikon.gabler.de>

## Part III

# BlueJ

Wie schon in der Einleitung kurz angesprochen ist BlueJ eine Entwicklungsumgebung für die Programmiersprache Java, "die speziell für den Schulunterricht entworfen wurde."<sup>3</sup> BlueJ bietet dem Benutzer neben der objektorientierten Modellierung unter anderem die Anzeige eines UML-Klassendiagramms, welches nach dem Erzeugen der Objekte die Beziehungen untereinander verdeutlicht.<sup>4</sup> Der Nutzer hat bei BlueJ die Möglichkeit, bereits vorgefertigte Klassen mit vorgefertigten Diensten zu verwenden oder neue Klassen zu implementieren, die auf die bereits vorhandenen Klassen aufbauen. Auf dieser Grundlage ist das Fangspiel, welches in dieser Facharbeit thematisiert wird, entstanden.

## Part IV

# Fangspiel

### 1 Grundaufbau

#### 1.1 Spielkonzept

Das Fangspiel besteht aus einem grauen Läufer, der von dem Spieler gesteuert wird und zwei roten Fängern, die den Spieler fangen sollen. Sowohl der Läufer als auch einer der beiden Fänger befinden sich von Beginn an auf dem Spielfeld. Wenn man nach 15 Sekunden noch nicht gefangen wurde, kommt der zweite Fänger ins Spiel. Des weiteren folgen nach jeweils 15 weiteren Sekunden Geschwindigkeitserhöhungen der Fänger. Beide können bei einem gewissen Abstand den Läufer "erkennen" und sich zu ihm bewegen. Außerdem können sie sich von der Wand abwenden, wenn sie sich nah an dieser befinden und sich abhängig von der Position des Läufers neu orientieren. Sie wenden sich auch voneinander ab, wenn der Abstand zwischen ihnen zu niedrig ist. Dabei gelten für beide Fänger die gleichen Bedingungen. Sobald einer der Fänger den Läufer fängt, der Läufer gegen die Wand läuft oder der Spieler die Maus doppelt klickt, ist das Spiel damit beendet.

#### 1.2 Klasse "Läufer"

Die Klasse "Läufer" wird mit der vorgefertigten Klasse "Buntstift" aus der SuM-Bibliothek erzeugt und beinhaltet die drei Zustände "zHPos", "zVPos" und "zGeschw". "zHPos" und "zVPos" sind für die Abfragen der Position des Läufers und "zGeschw" für die Geschwindigkeit des Fängers beim Laufen notwendig. Beim Erzeugen eines Läufers müssen Werte für die Parameter "pFarbe", "pHPos", "pVPos" und "pGeschw" angegeben werden. Nachdem die Dienste des Stiftes zum Zeichnen des Fängers benutzt werden, werden den drei Variablen jeweils die Werte der Parameter übergeben und damit gespeichert. Der Läufer kann bestimmte Dienste ausführen. Er kann seine horizontale sowie seine vertikale Position zurückgeben, laufen, sich nach links, rechts, oben oder unten drehen und freigegeben werden, wenn er nicht mehr benötigt wird.

---

<sup>3</sup>Kölling, Michael - <http://www.bluej.org/tutorial/blueJ-tutorial-deutsch.pdf>

<sup>4</sup>Kölling, Michael - <http://www.bluej.org/about/what.html>

### 1.3 Klasse "Fänger"

Die Klasse "Fänger" weist einige Parallelen zu der Klasse "Läufer" auf, denn auch der Fänger wird mit der Klasse "Buntstift" erzeugt und beinhaltet die Zustände und Dienste, die der Läufer besitzt. Allerdings kann der Läufer einige zusätzliche Dienste ausführen, die ein Läufer nicht kann. Er kann sich bis zu einem bestimmten Punkt oder um einen gewissen Wert bewegen sowie um einen gewissen Wert schneller beim Laufen werden. Hier ein Beispiel für die Anfänge zum Implementieren dieser Klasse:

```
public class Fänger
{
    Buntstift meinStift = new Buntstift();
    double zHPos;
    double zVPos;
    double zGeschw;

    public Fänger(double pHPos, double pVPos, double pGeschw, int pFarbe)
    {
        meinStift.bewegeBis(pHPos,pVPos);
        meinStift.setzeFarbe(pFarbe);
        meinStift.setzeFuellMuster(1);
        meinStift.zeichneRechteck(25,25);
        zHPos = pHPos;
        zVPos = pVPos;
        zGeschw = pGeschw;  }
```

### 1.4 Anwendungsklasse "Spiel"

Die Klasse "Spiel" ist die Anwendungsklasse des Fangspiels, wo die Klassen "Läufer" und "Fänger" initialisiert werden. Neben den beiden Klassen werden noch die Klassen "Bildschirm", "Maus", "Tastatur", "Buntstift" und "Uhr" benötigt. Wie in den zwei zuvor genannten Klassen werden auch hier Zustandsvariablen eingebaut. "zLäuferGeschw" und "zFängerGeschw" werden als Parameter für die Geschwindigkeit des Läufers beziehungsweise der Fänger verwendet, während sowohl die Variablen i, j und k als auch die Zustände "zIstGefangen", "zGegenDieWand", "zFängerEinsIstAktiv" und "zFängerZweIstAktiv" für Bedingungen der regelbasierten Abläufe benutzt werden, die im Punkt 4.2 genauer erläutert werden. Nachdem alle Klassen initialisiert und deklariert wurden, wird zuerst das Spielfeld von dem Buntstift gezeichnet:

```
derBuntstift.setzeFuellMuster(1);
derBuntstift.zeichneRechteck(1000,50);
derBuntstift.zeichneRechteck(50,750);
derBuntstift.bewegeBis(950,0);
derBuntstift.zeichneRechteck(50,750);
derBuntstift.bewegeBis(0,700);
derBuntstift.zeichneRechteck(1000,50);
```

Danach wird die Uhr gestartet, damit während dem Ausführen des Programms die verstrichene Zeit gezählt wird und für diverse Bedingungen verwendet werden kann. Mit der darauffolgenden Schleife wird dafür gesorgt, dass solange die Maus nicht doppelt geklickt, der Läufer nicht gefangen wurde oder gegen die Wand gelaufen ist, weitere Dienste ausgeführt werden können. Wenn diese Bedingung nicht mehr erfüllt ist, ist die Schleife beendet, die Uhr gestoppt und das Programm gibt die Spielzeit in Millisekunden mit einem erzeugten Text in der Konsole wieder. Als letztes werden alle Klassen wieder freigegeben und das Spiel beendet.

## 2 Regelbasierte Abläufe

### 2.1 "Die Wand erkennen"

Der erste regelbasierte Ablauf der künstlichen Intelligenz ist das Erkennen der Wand. Hierfür muss der Fänger, wenn "zFängerEinsIstAktiv" beziehungsweise "zFängerZweIstAktiv" wahr ist, wissen, bei welchen horizontalen oder vertikalen Positionen er umkehren muss. Aufgrund der Begrenzung des Spielfelds, die auf jeder Seite 50 Pixel beansprucht und der Position des Rechtecks von der oberen linken Ecke ausgehend, muss entweder ab einer horizontalen Position von 51 oder 924 Pixel oder bei einer vertikalen Position von 51 oder 674 Pixel umgekehrt werden, da sonst die Begrenzung des Spielfelds "überzeichnet" wird. Sobald der Fänger umgekehrt ist, wird die Differenz zu der horizontalen und der vertikalen Position des Läufers berechnet und sich entscheidend nach dem Ergebnis nach links, rechts, oben oder unten gedreht. Das ist der Quelltext für das Drehen zu einer Seite:

```
if(derLäufer.gibHPos() - derFänger.gibHPos() >=
    derLäufer.gibVPos() - derFänger.gibVPos())
{
    if(derLäufer.gibHPos() <= derFänger.gibHPos())
    {
        derFänger.nachLinks();
    }
    else
    {
        derFänger.nachRechts();
    }
}
else
{
    if(derLäufer.gibVPos() <= derFänger.gibVPos())
    {
        derFänger.nachOben();
    }
    else
    {
        derFänger.nachUnten();
    }
}
}
```

## 2.2 "Den Läufer sichten"

Der zweite regelbasierte Ablauf ist das Sichten des Läufers. Wie beim "Drehen zu einer Seite" werden auch hier die horizontale und vertikale Position des Läufers und des Fängers benötigt. Es gibt vier Fälle, die auftreten können. 1: Die horizontale Position des Fängers ist um höchstens 600 Pixel kleiner als die des Läufers.

```
if(derLäufer.gibHPos() - derFänger.gibHPos() >= -600 &&  
    derLäufer.gibHPos() - derFänger.gibHPos() < 0)  
{  
    derFänger.nachLinks();  
    derFänger.laufe();  
}
```

2: Die horizontale Position des Fängers ist um höchstens 600 Pixel größer als die des Läufers.

```
if(derLäufer.gibHPos() - derFänger.gibHPos() <= 600 &&  
    derLäufer.gibHPos() - derFänger.gibHPos() > 0)  
{  
    derFänger.nachRechts();  
    derFänger.laufe();  
}
```

3: Die vertikale Position des Fängers ist um höchstens 450 Pixel kleiner als die des Läufers.

```
if(derLäufer.gibVPos() - derFänger.gibVPos() >= -450 &&  
    derLäufer.gibVPos() - derFänger.gibVPos() < 0)  
{  
    derFänger.nachOben();  
    derFänger.laufe();  
}
```

4: Die vertikale Position des Fängers ist um höchstens 450 Pixel größer als die des Läufers.

```
if(derLäufer.gibVPos() - derFänger.gibVPos() <= 450 &&  
    derLäufer.gibVPos() - derFänger.gibVPos() > 0)  
{  
    derFänger.nachUnten();  
    derFänger.laufe();  
}
```

Der Laufe-Dienst, der nach dem Drehen des Fängers aufgerufen wird, sorgt dafür, dass der Fänger sich diagonal bewegen kann und es somit schwerer für den Läufer ist, nicht gefangen zu werden.

## 2.3 "Zweiten Fänger auf das Spielfeld bringen"

Der dritte regelbasierte Ablauf ist das "auf das Spielfeld bringen" des zweiten Fängers. Der wurde zwar schon von Anfang an initialisiert, allerdings auf einer nicht auf dem erzeugten Bildschirm sichtbaren Position. Das "Erzeugen" des zweiten Fängers muss auf diesem Weg durchgeführt werden, da die Initialisierung einer Klasse weder nach dem Drücken der Tastatur noch direkt nach dem Ablaufen einer bestimmten Zeit geschehen kann. Das Ausweichen auf diesen Lösungsweg gelingt durch den Zustand "zFängerZweilstAktiv" und der Wert der Variable "i". Nachdem die verstrichene Zeit im Programm 15 Sekunden beträgt, trifft "zFängerZweilstAktiv" zu und die Variable "i" wird mit Eins addiert, damit die Bedingung nicht dauerhaft zutreffen kann. Es wird ebenfalls ein Text in der Konsole ausgegeben, der auf das aktuelle Spielgeschehen hinweist. So sehen die dazugehörigen Dienste aus:

```
if(dieUhr.verstricheneZeit() > 15000 && j == 0)
{
    j++;
    zFängerZweilstAktiv = true;
    System.out.println("Ein Fänger ist anscheinend zu einfach für dich...");
}
```

```
if(zFängerZweilstAktiv == true)
{
    derFänger2.laufe();

    [...]

}
```

Sobald sich der zweite Fänger auf dem Spielfeld befindet, gelten für ihn die gleichen Bedingungen wie für den ersten Fänger. Wenn die Bedingungen für den zweiten Fänger schon von Anfang an gegolten hätten, wäre er über die Spielfeldbegrenzung hinaus gelaufen und somit frühzeitig auf das Spielfeld gelangt.

## 2.4 "Die Fänger nicht aneinander geraten lassen"

Dieser Ablauf beschäftigt sich wieder mit der horizontalen und vertikalen Positionen der Fänger, da sie sich nicht selbst treffen oder "überlaufen" sollen. Wie auch bei dem Erkennen der Wand und dem Sichten des Läufers wird hier mit Positionsabfragen gearbeitet, bis die Rechenoperationen im Bereich von zwei Werten sind. In diesem Fall müssen für eine Bedingung vier Rechenaufträge mit den Positionsabfragen erfüllt sein. Ähnlich wie beim Sichten des Läufers gibt es vier verschiedene Fälle:

```

if(derFänger.gibHPos() - derFänger2.gibHPos() <= 150 &&
   derFänger.gibHPos() - derFänger2.gibHPos() > 0 &&
   derFänger.gibVPos() - derFänger2.gibVPos() <= 100 &&
   derFänger.gibVPos() - derFänger2.gibVPos() > 0      )
{
    derFänger.nachUnten();
    derFänger.laufe();
    derFänger2.nachLinks();
    derFänger.laufe();
}

```

```

if(derFänger.gibHPos() - derFänger2.gibHPos() <= 150 &&
   derFänger.gibHPos() - derFänger2.gibHPos() > 0 &&
   derFänger.gibVPos() - derFänger2.gibVPos() >= -100 &&
   derFänger.gibVPos() - derFänger2.gibVPos() < 0      )
{
    derFänger.nachRechts();
    derFänger.laufe();
    derFänger2.nachUnten();
    derFänger2.laufe();
}

```

```

if(derFänger.gibHPos() - derFänger2.gibHPos() >= -150 &&
   derFänger.gibHPos() - derFänger2.gibHPos() < 0 &&
   derFänger.gibVPos() - derFänger2.gibVPos() <= 100 &&
   derFänger.gibVPos() - derFänger2.gibVPos() > 0      )
{
    derFänger.nachRechts();
    derFänger.laufe();
    derFänger2.nachOben();
    derFänger2.laufe();
}

```

```

if(derFänger.gibHPos() - derFänger2.gibHPos() >= -150 &&
   derFänger.gibHPos() - derFänger2.gibHPos() <= 0 &&
   derFänger.gibVPos() - derFänger2.gibVPos() >= -100 &&
   derFänger.gibVPos() - derFänger2.gibVPos() <= 0      )
{
    derFänger.nachOben();
    derFänger.laufe();
    derFänger2.nachLinks();
    derFänger2.laufe();
}

```

Die Fänger laufen jeweils in die Richtung, die dafür sorgt, sich aus der zutreffenden Bedingung zu "lösen".



## 2.5 "Die Geschwindigkeit der Fänger erhöhen"

Die erste Erhöhung der Geschwindigkeit tritt ab einer verstrichenen Zeit von 30 Sekunden auf und wird stets nach 15 weiteren Sekunden wiederholt. Dabei erlangt der zweite Fänger gegenüber dem ersten Fänger eine höhere Geschwindigkeit. Diese Bedingungen werden, ähnlich wie bei dem "Erzeugen" des zweiten Fängers, mit Variablen versehen, da bei der verstrichenen Zeit der Uhr kleinere Unregelmäßigkeiten entstehen, die eine genaue zeitliche Eingrenzung "übergehen" könnten. Bis zu einer Spielzeit von 90 Sekunden wird für jede zwischenzeitliche Erhöhung eine eigene Bedingung erfüllt, da in der Konsole verschiedene Texte vom System ausgegeben werden sollen. Als Grundprinzip lassen sich diese Operationen verdeutlichen:

```
{
    j++;
    derFänger.werdeSchneller(0.0125);
    derFänger2.werdeSchneller(0.025);
}
```

Mit der 105. Sekunde gibt es nur noch eine Bedingung, die stets den gleichen Text ausgibt. Hier wird anstatt einer vorher eingegebenen Zahl in Millisekunden mit der Variable k gearbeitet:

```
if(dieUhr.verstricheneZeit() > k)
{
    k = k + 15000;
    derFänger.werdeSchneller(0.25);
    derFänger2.werdeSchneller(0.5);
    System.out.println("...");
}
```

## 2.6 "Der Läufer wurde gefangen"

Der letzte Unterpunkt der regelbasierten Abläufe beschäftigt damit, wann der Läufer vom Fänger gefangen wurde. Hier wird erneut mit dem Abfragen der Positionen von Läufer und Fänger und zusätzlich mit dem Zustand "zIstGefangen" gearbeitet. Wenn der Abstand zwischen dem Läufer und dem Fänger sowohl horizontal als auch vertikal unter 25 Pixeln beträgt, ist "zIstGefangen" wahr, was zum Abschluss der Schleife führt. Das Spiel ist vorbei und die Uhr gestoppt. Beispiel für eine Bedingung:

```
if(derLäufer.gibHPos() - derFänger.gibHPos() < 25 &&
    derLäufer.gibHPos() - derFänger.gibHPos() > 0 &&
    derLäufer.gibVPos() - derFänger.gibVPos() < 25 &&
    derLäufer.gibVPos() - derFänger.gibVPos() > 0 )
{
    zIstGefangen = true;
}
```

## Part V

# Fazit

An dieser Stelle möchte ich mich auf meine in der Einleitung genannten Intention beziehen, das Thema "Künstliche Intelligenz in Computerspielen - Regelbasierte Abläufe veranschaulicht an einem Spiel" zu behandeln. Mir wurde schnell deutlich, wie viele Bedingungen allein erfüllt sein müssen, damit beispielsweise der Fänger dem Läufer folgen kann. Ich habe auch schnell gemerkt, dass nicht alle Ideen, die man sich ausgedacht hat, ohne weiteres umsetzbar sind. Nachdem ich mich dazu entschieden hatte, einen zweiten Fänger zu integrieren, hatte ich doch einige Schwierigkeiten damit, diesen in das Fangspiel einzubinden. Am Ende ist es mir doch gelungen, was mir die Notwendigkeit für das sorgfältige Planen und der nötigen Zeit der Implementierung verdeutlicht. Selbst "Künstliche Intelligenz" für ein kleines Fangspiel zu programmieren ist ein sehr zeitaufwändiges Thema, womit man sich lange und ausgiebig beschäftigen muss. Zwar steht dieses Fangspiel in keiner Relation zu beispielsweise einem Rennspiel, allerdings habe ich trotzdem überzeugende Eindrücke davon bekommen, wie aufwändig und sorgfältig "künstliche Intelligenz" erzeugt werden muss.

## Part VI

# Anhang

<https://dl.dropbox.com/u/40837787/Fangspiel.zip>

## Part VII

# Quellenverzeichnis

- [1] Kern, Sabine - 02.06.2006 - [http://www.planet-wissen.de/\[...\]](http://www.planet-wissen.de/[...])  
[2] Prof. Dr. Richard Lackes, Dr. Markus Siepermann - <http://wirtschaftslexikon.gabler.de/Definition/kuenstliche-intelligenz-ki.html> [3] <http://www.bluej.org/tutorial/blueJ-tutorial-deutsch.pdf> [4] <http://www.bluej.org/about/what.html>

# Selbstständigkeitserklärung:

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benutzt habe. Die Stellen der Facharbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Das Gleiche gilt auch für beigegebene Zeichnungen, Kartenskizzen und Darstellungen.