View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by ePublications

Fachhochschule Köln
University of Applied Sciences Cologne
Abteilung Gummersbach

Fachbereich Informatik

Diplomarbeit

Zur Erlangung
des Diplomgrades
Diplom- Informatiker (FH)
in der Fachrichtung Informatik

Thema

- Charakterverhalten in Computerspielen: Pathfinding –

Erstprüfer:	Prof. Dr. H. Faeskorn – Woyke
Zweitprüfer:	Prof. Dr. E. Ehses
vorgelegt von	René Greulich
aus	Georgstr. 3
	51145 Köln
Tel.- Nr:	02203/ 23026
Email:	geek@geekweb.de
Matr. – Nr:	110 18781

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Inhaltsverzeichnis

1.0 Vorwort 6

Ziel der Diplomarbeit 6

2.0 Artificial Intelligence (AI – Künstliche Intelligenz) in Computerspielen.. 9

Ein genereller Überblick 9

Einleitung 9

Anwendungsgebiete 10

Stand und Zukunft der AI - Technologien 13

AI in der Spieleprogrammierung 15

3.0 AI in Bezug auf das Verhalten von Charakteren in Computerspielen..... 18

Charaktere 18

Lokomotion – Motor Skills 20

Steering 22

Steuerung von Entitäten 22

Motivation – Action Steering 25

Entscheidungsfindung 25

4.0 Steering 36

Pathfinding- Algorithmen 36

Einleitung 36

Algorithmen, basierend auf der Ignorierung des Hindernisses 37

Algorithmen, auf Graphen basierend 41

Heuristische Suchalgorithmen, auf Graphen basierend 46

5.0 Pathfinding in 2D Welten 48

Die Graphentheorie 48

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Erläuterung 48
A*- Algorithmus 49
Grundlagen 49
Erläuterung des A* 51
Detailerläuterung 55

6.0 Optimierung des A* 62

Optimierung 62
Ästhetische Optimierung 62
Ziel der ästhetischen Optimierung 62
Straight Paths – Ein gerader Weg 63
Smooth Paths – Pfade mit geschmeidiger Kurvenführung 64
Hierarchische Pfadsuche 67
Fazit 70
Speicher-/ Performanz-/ Geschwindigkeitsoptimierung 72
Optimierung des A* 72
Suchbereichsoptimierung - Erstellung der minimalen Suchmenge 74
Optimierung des Algorithmus 79
Fazit 85

7.0 Praktische Umsetzung des Pathfindings 86

Voraussetzungen 86
Entscheidungen 86
Spezifikation 87
Allgemein 88
Die phänomenale Gestalt – Teil 1 94
Lokomotion & Steering 94
Umsetzung der A* Basis 96
Areas · Nachbarknoten · Grids · Datentypen · Knotenbank · Heuristik .. 96
Forced Portal A* Search - Das neue Pathfindingkonzept 118

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Die A* Erkenntnis 118
Forced Portal A* Search 126
Pure Portal To Portal Search (PPTP) 131
Portal To Portal Search (PTP) 134
Last Portal Search (LPS) 141
Goal Area Search (GAS) 144
Direkte Pfade 145
Fazit 145
Pathfindingsystem 146
Implementierung des A* 146
Die phänomenale Gestalt – Teil 2 157
Motivation 157
Voraussetzungen und Verbesserungen 163

8.0 Resümee 167
Die phänomenale Gestalt in Computerspielen 167

9.0 Literaturverzeichnis 172
Internet 172
Bücher 173
Zeitschriften 174
Dokumentationen 174
Aufsätze 174

10.0 Anhang 176
Debug- Ausgabe 176

11.0 Index 198

Autor: René Greulich Matrikelnr.: 110 18781
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke Zweitprüfer: Prof. Dr. E. Ehses

Abbildungsverzeichnis

Abbildung 01: AI Ressourcen Umfrage auf der Game Developer Conference 2001... 16
Abbildung 02: Charaktere ... 18
Abbildung 03: First Person View ... 20
Abbildung 04: Third Person View ... 20
Abbildung 05: Ausweichmanöver notwendig ... 23
Abbildung 06: Aufgrund der U-Form versagt der Algorithmus ... 24
Abbildung 07: Theoriematrix ... 29
Abbildung 08: belief/ desire/intention- architecture ... 31
Abbildung 09: Hindernisumgehung bei kleinen Hindernissen durch Verschiebung .. 38
Abbildung 10: Endlossuche ... 38
Abbildung 11: Umrissverfolgung ... 39
Abbildung 12: Endlose Umrissverfolgung ... 39
Abbildung 13: kontrollierte Umrissverfolgung ... 40
Abbildung 14: Möglichkeiten der Aufteilung einer Lokation ... 56
Abbildung 15: Nachbar- /Nachfolge-/ Kindknoten ... 59
Abbildung 16: Vergleich normaler A* Pfad (oben) und gerader A* Pfad (unten) ... 63
Abbildung 17: normaler und geschmeidiger Pfad ... 67
Abbildung 18: Geschmeidiger Pfad anhand von Bézier Kurven ... 67
Abbildung 19: Genereller Weg ... 69
Abbildung 20: Genereller und idealer Weg ... 70
Abbildung 21: Rectangular Grid ... 75
Abbildung 22: Actual polygonal floor ... 76
Abbildung 23: Polygonal floor representation ... 77
Abbildung 24: Points of visibility ... 78
Abbildung 25: Suchausbreitung abhängig von der Heuristik ... 80
Abbildung 26: Konkaves Hindernis ... 81
Abbildung 27: Unästhetisches Ergebnis der normalen A* Suche ... 89
Abbildung 28: Laufareas (rote Linien), es werden keine Pfade verwendet ... 97
Abbildung 29: Auszug einer Lokationgrid ... 98
Abbildung 30: Unästhetisches Ergebnis der normalen A* Suche ... 118
Abbildung 31: Genereller und idealer Weg bei der hierarchischen Pfadsuche ... 122
Abbildung 32: Unästhetisches Ergebnis der normalen A* Suche ... 127
Abbildung 33: besseres Ergebnis mit hierarchischer A* Suche ... 128
Abbildung 34: schlechtes Ergebnis mit hierarchischer A* Suche ... 129
Abbildung 35: Lösung: forced portal A* Suche ... 130
Abbildung 36: Beispiel forced portal A* Suche ... 131
Abbildung 37: Wahl des Mittelpunktes zwischen Portal und Zielpunkt ... 141
Abbildung 38: Mittelpunkte zwischen Portal und Zielpunkt (Projektion) ... 142
Abbildung 39: Regelmäßige Vielecke ... 164

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

1.0

Vorwort

Ziel der Diplomarbeit

Die Bedeutung der *künstlichen Intelligenz (KI)* in Computerspielen hat sich in den letzten Jahren bedeutend geändert. Es wird versucht Spielfiguren zu implementieren, die ein reales, menschenähnliches Verhalten aufweisen. Der primäre Grund ist die Erkenntnis, dass eine gute KI von *Charakteren*[1] zum besseren Verkauf von Computerspielen beiträgt. Die Käufer wünschen reale Charaktere, mit denen sie sich identifizieren können. Der sekundäre Grund ist die Verbesserung der Computertechnik zugunsten einer Entlastung der CPU, die eine gute KI erst ermöglicht hat. Zurückzuführen ist dies auf die Optimierung der CPUs und des Arbeitsspeichers und vor allem aufgrund der Entwicklung leistungsstarker, moderner 3D Graphikkarten. Gerade diese erlauben mehr CPU-Rechenzeit für „*Nicht-Grafiktasks*[2]“.

Aufgrund des oftmals unrealistischen Charakterverhaltens in Computerspielen ist das Ziel der Diplomarbeit, die Entwicklung einer Spielfigur, im Sinne einer

[1] *Charaktere:* Spielfiguren

[2] *Nicht- Grafiktasks*: Bei einem Task handelt es sich um eine Aufgabe, die die *Engine* (die Ansammlung aller c/c++ Bibliotheken, die die Funktionalität eines Spiels ausmacht) zu erledigen hat. Eine Spieleengine umfasst in der Regel nur einen Thread, d.h. Engines sind meistens keine Multithreading- Systeme. Ein Thread ist die kleinste Ausführungseinheit, die von Win32 geplant werden kann. Innerhalb eines Threaddurchlaufs werden eine Vielzahl von Tasks bearbeitet. Prinzipiell gibt es zwei große Taskbereiche, den Graphikbereich und den Spiellogikbereich. Die Grafiktasks sorgen dafür, dass die Graphiken geblittet (2D) oder gerendert (3D) werden. Die Spiellogiktasks sorgen dafür, dass alle anderen spielspezifischen Arbeiten erledigt werden. Hierunter fallen auch die Aufgaben der künstlichen Intelligenz. Wenn eine Spielfigur ihren Weg berechnen soll, dann wird ein Task „Berechne deinen Weg“ gestartet.

phänomenalen Gestalt[3]. Die *phänomenale Gestalt* entsteht in dem Moment, indem ein Objekt oder ein Charakter reales Leben dermaßen echt abbildet, dass man ein „künstliches" Bewusstsein wahrnehmen kann. Die Wahrnehmung findet statt, weil der Charakter fähig ist, sein Bewusstsein, sein Verhalten und das Verhalten anderer zu reflektieren.

Die künstliche Intelligenz von Spielfiguren besteht aus den Bereichen *Lokomotion*, *Steering* und *Motivation*. Das Ziel, im Hinblick auf die *phänomenale Gestalt*, ist die perfekte Abbildung realistischen Verhaltens hinsichtlich dieser Bereiche. Dementsprechend stellt sich im Bereich des *Lokomotions*, welcher die Animationsverwaltung des Charakters darstellt, die Aufgabe, zu jedem Zeitpunkt die richtige Bewegung des Charakters auszuführen. Es muss die Redeanimation gespielt werden, wenn der Charakter sprechen soll, die Laufanimation, wenn er laufen soll, etc. Das *Steering* steuert die Pfadfindung eines Charakters. Das Ziel des *Steerings* ist die Generierung eines realistischen Laufpfades, bei dem beispielsweise vermieden wird, dass der Charakter durch Wände geht oder reißende Flüsse durchwandert. Die perfekte *Motivation* muss das Verhalten des Charakters hinsichtlich seiner Entscheidungsfindung steuern. Bei der Entscheidungsfindung handelt es sich um die Fähigkeit der Spielfigur, situationsbedingt Entscheidungen gemäß seines Charakters zu treffen.

Die Aufgabe der Diplomarbeit ist, einen generellen Überblick über das *Lokomotion* und die *Motivation* zu verschaffen. Primäre Aufgabe ist die Umsetzung des *Steerings*.

Prinzipiell umfasst beinahe jedes Genrespiel ein Pathfindingsystem und fast alle Charaktere finden auch einen Weg von A nach B. Das Problem vieler Computerspiele ist, dass das Pathfinding in ästhetischer Sicht meist unrealistisch

[3] *phänomenale Gestalt*: Der Begriff wurde zur Vereinfachung für die Diplomarbeit entwickelt. Hierbei handelt es sich um ein Objekt mit phänomenalem Gehalt, um ein Objekt mit künstlichem Bewusstsein. Der phänomenale Gehalt ist ein entliehener Begriff der Philosophie u.a. zur Untersuchung des postbiotischen Bewusstseins. Diese Begrifflichkeit, auf die noch weiter eingegangen werden wird, wird von Thomas Metzinger verwendet.

umgesetzt wird. Folgt ein Mensch beispielsweise einem kurvenförmigen Hindernis, dann läuft er meist entsprechend des Kurvenverlaufs. Im Gegensatz dazu entwickeln Spielfiguren meist einen sehr eckigen, unrealistischen Weg, was dazu führt, dass die Fortbewegung der Charaktere sehr abgehackt aussieht. Ziel dieser Diplomarbeit ist eine Möglichkeit aufzuzeigen, nachdem die Basispfadfindung erläutert wurde, eine ästhetische Pfadfindung zu implementieren, um dem Charakter als *phänomenaler Gestalt* näher zu kommen, wobei auch die Performanz und die Speicherverwaltung betrachtet wird. Des weiteren soll gezeigt werden, wie eine *Motivation* hinsichtlich der Pfadfindung aussehen kann.

Zusammengefasst werden folgende Fragen geklärt:

1. Was ist die künstliche Intelligenz ?
2. Was ist die künstliche Intelligenz von Charakteren in Computerspielen (*Lokomotion, Steering, Motivation*) ?
3. Mit welchen Pathfindingmethoden kann das Steering umgesetzt werden ?
4. Wie verbessert man die Pathfindingmethode, um ein realistisches Ergebnis zu erreichen (Umsetzung dieser Pathfindingmethode in einem Beispielprogramm) ? selbstentwickelte Lösung: *forced portal A* search.*
5. Wie sieht eine Motivation anhand der *forced portal A** Suche aus ?

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

2.0

Artificial Intelligence (AI – Künstliche Intelligenz) in Computerspielen

Ein genereller Überblick

Einleitung

Bei der künstlichen Intelligenz handelt es sich um ein interdisziplinäres Forschungsgebiet, bestehend aus den Disziplinen Informatik, Psychologie, Biologie, Linguistik und Mathematik. Entstanden ist die KI in den fünfziger Jahren in den USA unter der Bezeichnung *Artificial Intelligence (AI)*. Die Artificial Intelligence verfolgt das Ziel der Beschreibung und Erklärung der Funktionsweise, sowie der künstlichen Nachbildung des menschlichen Gehirns, anhand von Computern. Ein weiteres Ziel ist die Verbesserung der Arbeitsweise von Computerprogrammen durch die Nachbildung menschlicher Problemlösungsansätze. Das Endziel der künstlichen Intelligenz ist die Entwicklung eines menschenähnlichen Roboters bzw. menschenähnlicher Systeme.

Bekanntestes Beispiel für intelligente Maschinen ist der IBM- Schachcomputer *Deep Blue*, der 1997 den Schachweltmeister Garri Kasparow, mit zwei Siegen, einer Niederlage und drei Remis schlug. Ein weiteres ist *Sojourner*, der Mars Pathfinder Rover der Nasa, der im September 1997 den Mars erkundete.

Um festzustellen, wann einer Maschine Intelligenz zugesprochen werden kann, erdachte sich der englische Mathematiker Alan M. Turing 1950 den *Turing Test*. Bei

diesem Test hat ein menschlicher Gutachter die Möglichkeit, gleichzeitig einer Maschine und einem Menschen, im Sinne eines Dialoges, Fragen zu stellen, ohne zu wissen, wer der Mensch und wer die Maschine ist. Eine Maschine gilt bei diesem Test als intelligent, wenn der Gutachter nicht herausfinden kann, wer die Maschine ist. Deep Blue ist eine der wenigen intelligenten Maschinen, die den Test bestehen würden. Kasparow hätte, während einer Partie gegen Deep Blue und einem Kollegen, keine Möglichkeit herauszufinden, wer der Computer ist. Gegner dieser Theorie, wie Dorn[4] und Gottlob[5], argumentieren, dass eine Maschine wie Deep Blue den Turing Test nicht bestehen kann, weil mit diesen Maschinen kein klassischer Dialog möglich ist. Befürworter wie Helge Ritter[6] sind der Meinung, dass der Turing Test auf dem Gebiet des Schachspiels als gelöst gilt.

Anwendungsgebiete

Prinzipiell unterteilt sich die künstliche Intelligenz in die Gebiete *Maschinelles Beweisen, Spieleprogrammierung, Verarbeitung natürlicher Sprache, Expertensysteme, Robotik* und *Neuronale Netze*.

Maschinelles Beweisen

Das maschinelle Beweisen ist, geschichtlich gesehen, die älteste Disziplin der Artificial Intelligence. Hierbei wird bewiesen, indem vorliegendes formales Wissen mit intelligenten Schlussfolgerungen verarbeitet wird.

Spielprogrammierung

Die Spieleprogrammierung ist das zweite Teilgebiet der künstlichen Intelligenz. Sie unterteilt sich in die Entwicklung von Spielen, die Lösungsstrategien verfolgen, um ein Grundproblem zu lösen, wie z.B. Schach oder Scrabble, und in das Gebiet der Genrespiele wie Shooter, Adventure, Sport-, Strategie- und Jump and Run-Spiele. Schon 1946 begann Turing mit der Entwicklung des ersten Schachcomputers

[4] *Dorn:* Juergen; Prof. Dipl.-Inf. Dr. Ing. (Technischen Universität Wien)

[5] *Gottlob:* Georg; Prof. Dipl.-Ing. Dr. techn. (Technischen Universität Wien)

[6] *Helge Ritter:* Professor an der Technischen Fakultät Bielefeld, Leiter der Arbeitsgruppe Neuroinformatik

auf einer aufgerüsteten Dechiffriermaschine. Die Genrespiele sind im Gegensatz dazu noch relativ jung.

Verarbeitung natürlicher Sprache

Die Verarbeitung natürlicher Sprache ist eine der umfangreichsten Bereiche der künstlichen Intelligenz. Gerade im Sinne der Verbesserung der *Mensch-Computer- Interaktion (MCI)* im Multimediazeitalter ist die Bedeutung dieses Wissenschaftszweiges sehr groß.

Einsatzmöglichkeiten für die Verarbeitung von Sprache sind z.B. Übersetzungsprogramme, sprachliche Schnittstellen zu Informationssystemen, die Unterstützung behinderter Menschen im Umgang mit dem Personal Computer, usw.

Eines der ersten Programme in diesem Bereich war *ELIZA*, ein Programm, das einen Psychiater simulierte. Dieses Programm weckte damals große Erwartungen, die bis heute nicht erfüllt werden konnten. Das Problem von ELIZA war, dass es nur anhand der Syntax von Sprache agierte und den Zusammenhang der Sätze nicht richtig verfolgen konnte. Heute ist man sich bewusst, wie wichtig der Zusammenhang von Worten im Kontext ist.

Expertensysteme

Expertensysteme wurden entwickelt, um das beste Wissen der besten Experten zeit- und raumlos zu machen; um es jederzeit jedermann zur Verfügung zu stellen. Prinzipiell wird bei Expertensystemen Wissen anhand von gespeichertem Erfahrungswissen mittels heuristischen Regeln modelliert.

Robotik

Die Robotik ist wohl der bekannteste und umstrittenste Bereich der Artificial Intelligenz. Als Endziel wird versucht, den Menschen ganzheitlich abzubilden, was mitunter zu der utopischen Befürchtung führt, dass der Mensch durch den Roboter ersetzt werden könnte. Realistisch gesehen werden Roboter entwickelt, um Bilder zu erkennen, zur Kommunikation, um Sprache zu verstehen, zum Lernen, Wissen zu

speichern, logisch zu Denken, zu Laufen, das Greifen zu simulieren, usw. Roboter sind unfähig ein „menschliches" Bewusstsein zu entwickeln, Emotionen zu simulieren und kreativ zu sein. Eine interessante Frage der Robotik ist, wodurch ein nichtbiologisches, informationsverarbeitendes System zu einem Subjekt von Erfahrung wird und ob dies überhaupt möglich ist.

Die Robotik betreffend gibt es in der Philosophie, wie schon erwähnt, eine Untersuchung von Prof. Dr. Thomas Metzinger bezüglich des „*phänomenalen Gehaltes*"[7] eines Objektes. Eine *phänomenale Gestalt* entsteht, wenn das System ein phänomenales Bewusstsein, also „*phänomenales Gehalt*" hat, d.h., dass die Gestalt ein „künstliches" Bewusstsein bezüglich der Probleme ihrer Welt besitzt.

„Phänomenales Gehalt entsteht dann, wenn sich die repräsentationalen Zustände eines informationsverarbeitenden Systems für dieses selbst irgendwie anfühlt, also wenn sie einen introspektiv zugänglichen qualitativen Charakter besitzen". [8]

Prof. Dr. Thomas Metzinger[9]

Eine *phänomenale Gestalt* ist, vereinfacht ausgedrückt, ein „bewusstes Objekt".[10]

Prinzipiell ist uns bewusst, dass ein solches Verhalten von Robotern wahrscheinlich nie implementiert werden kann. In der Spieleprogrammierung aber ist gerade die *phänomenale Gestalt* dass, was die Qualität von Genrecomputerspielen in der heutigen Zeit, in der davon ausgegangen werden kann, dass die graphische Umsetzung hervorragend ist, auszeichnet. Das Faszinierende an der *phänomenalen Gestalt* ist, dass, auf einer emotionalen, menschenähnlichen Ebene, Charaktere eine, wenn auch virtuelle Realität erhalten, nachvollziehbare Charakterzüge entwickeln, sich plötzlich adäquat zu unserer Realität verhalten und bestimmten Verlangen folgen.

---

[7] Zitat: Metzinger, Computer.Gehirn Was kann der Mensch? Was können die Computer?, S.87

[8] Zitat: Metzinger, Computer.Gehirn Was kann der Mensch? Was können die Computer?, S.87

[9] *Thomas Metzinger:* Professor für Philosophie der Kognition an der Universität Osnabrück

[10] Zitat: Metzinger, Computer.Gehirn Was kann der Mensch? Was können die Computer?, S.87

Entsprechend dem Turing- Test, gibt es den *Metzinger- Test*, um festzustellen, wann ein Objekt eine *phänomenale Gestalt* ist. Ein Objekt ist dann eine phänomenale, reale, bewusste Gestalt, wenn es selbst über Bewusstsein nachzudenken beginnt.

Im Bereich der Robotik ist der Fortschritt künstlicher Intelligenz noch „relativ gering". Grund hierfür ist, dass erst die sensorischen Fähigkeiten, wie z.B. realistisches Greifen, entwickelt werden muss/ musste.

Neuronale Netze

Die bis hierhin aufgeführten Methoden formalisieren intelligentes Verhalten anhand von Symbolen. Menschliches Denken basiert aber auf physikalischen Prozessen. Mit Hilfe von neuronalen Netzen versucht man das Gehirn, welches auf Basis neuronaler Zusammenhänge funktioniert, zu erläutern. Die Forschung der neuronalen Netze begann in den fünfziger Jahren.

Stand und Zukunft der AI - Technologien

Lange Zeit galt die Wissenschaft der künstlichen Intelligenz als gescheitert. Nach den großen Erfolgen der ersten Jahre glaubte man die Entwicklung müsste in dieser Geschwindigkeit vorangehen, was allerdings nicht der Fall war. Das Problem, laut *Marvin L. Minsky*[11], ist, dass aufgrund des Turing- Testes falsche Erwartungen geweckt wurden. Es ist eine Tatsache, dass auf dem Gebiet der Verarbeitung von natürlicher Sprache noch nicht die erwarteten Fortschritte gemacht wurden. Man ist noch weit von dem Ziel entfernt, auf dem Gebiet des reinen Dialogs, einen Computer von einem Menschen zu unterscheiden. Das Problem der Spracherkennung ist, wie schon erwähnt, nicht die Sprache selbst, sondern vielmehr der zur Interpretation von Sprache notwendige „Menschenverstand". Einem Kind ist vollkommen bewusst, dass ein Spielzeugauto mit einem Seil gezogen werden kann, dass es aber unmöglich ist,

[11] *Marvin L. Minsky:* Marvin Minsky is Toshiba Professor of Media Arts and Sciences, and Professor of Electrical Engineering and Computer Science, at the Massachusetts Institute of Technology. He is one of the Pioneers of AI.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

dieses Auto mit dem Seil anzuschieben. Demselben Kind fallen spontan Hunderte von Dingen bzw. Situationen ein, wofür ein Seil verwendet und genauso viele, wofür es nicht verwendet werden kann. Einem Computer müssen alle diese Dinge „beigebracht“ werden. Wie aber muss eine Wissensbasis aussehen, um alle diese Informationen zu speichern und wie müssen die einzelnen verwendeten Datenstrukturen gestaltet werden ? Die Beantwortung dieser Frage ist auch heute noch nicht annähernd möglich. Ein weiteres Problem ist, dass ein einfaches Übersetzungsprogramm eine gewisse Vorstellung von der Psychologie des Menschen benötigt, um herausfinden zu können, was eine Person mit einer gewissen Phrase gemeint hat.

Laut Minsky wurden andererseits exzellente Erfolge in den Bereichen Expertensysteme, Mustererkennung, Robotik, etc., erzielt. Ein Beispiel für den Fortschritt sind die Experimente und Errungenschaften der Autoindustrie zur Entwicklung intelligenter Autos. Viele Bereiche des menschlichen Lebens werden heute von Computern und somit von KI beeinflusst: das Internet, der intelligente Kühlschrank, usw. Die Bedeutung moderner KI wird sich noch in vielen weiteren Bereichen menschlichen Lebens ausbreiten, sodass der weitere Aufstieg der künstlichen Intelligenz unabdinglich ist.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

AI in der Spieleprogrammierung

Wie bereits erwähnt, muss man einen Unterschied zwischen Spiele, die Lösungsstrategien verfolgen, wie z.B. Schach, und Genrespielen machen. Diese Diplomarbeit beschäftigt sich nur mit Genrespielen, wie z.B. Shooter, Adventure, Jump and Run, Sportspiele, usw. Die Artificial Intelligence dieser Spiele unterscheidet sich merklich von der „normalen" künstlichen Intelligenz, weil die Genrespiele nur eine gewisse künstliche Intelligenz simulieren wollen. Diese Aussage sollte nicht unterschätzt werden. Die Spiele AI versucht Realität so gut wie möglich abzubilden, allerdings darf nicht übersehen werden, dass dies hinsichtlich einer künstlichen Welt geschieht.

„Robotics in all its forms is perhaps the place where all of our "theoretical" AI knowledge gets put to the test in environments tougher than any game. After all, in the computer we have the luxury of being able to make up our physics as we go along; robotics are stuck dealing with real world issues that would drive most game AI developers batty."[12]

Steve M. Woodcock[13]

Der große Vorteil der Spiele AI ist, dass man, wie Woodcock erwähnt, in die Welt der Charaktere eingreifen und diese Welt im Sinne des Charakters verändern und somit die Voraussetzungen für ihr Verhalten verbessern kann.

Prinzipiell hat sich die Wissenschaft der künstlichen Intelligenz in Computerspielen in den letzen Jahren sehr stark verändert. Der Tenor der *Game Developer Conference (GDC) 2001* war, dass die Spiele AI nun endlich ein anerkannter Wissenschaftszweig der Spieleprogrammierung geworden ist. Grund

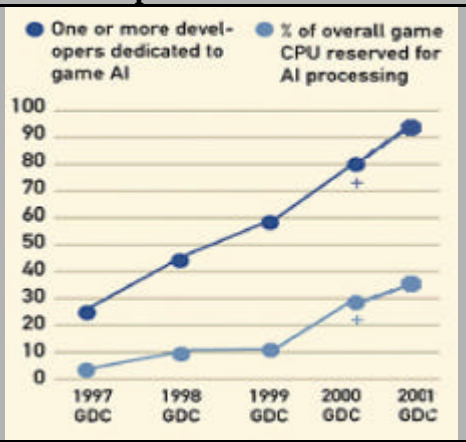
[12] Zitat: Woodcock, Game AI Resources: Robotics <www.gameai.com/robotics.html>

[13] *Steve M. Woodcock:* Chefprogrammierer beim National Missile Defense Battle Management Command, Control, and Communications project; außerdem ist er freier AI Spieleprogrammierer und einer der führenden Spezialisten für AI.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

hierfür ist die hervorragende künstliche Intelligenz des Spiels *The Sims*[14]. Die künstliche Intelligenz von The Sims hat, laut Woodcock, dafür gesorgt, dass es noch ein Jahr nach Erscheinung oben in den Verkaufscharts stand.

Abbildung 01:
AI Ressourcen Umfrage auf der Game Developer Conference 2001



In den sogenannten *roundtables*[15] der GDC wurde festgestellt, dass in 94% der Fälle mindestens ein Entwickler nur mit der Implementierung der Artificial Intelligence beschäftigt war. Weiterhin wurde darauf hingewiesen, dass die AI Entwickler erstmals genug CPU- Zeit zur Verfügung gestellt bekommen hatten (35 - 38% der gesamten CPU - Zeit), um eine gute künstliche Intelligenz umsetzen zu können. Wie schon erwähnt ist eine entscheidende Komponente für die höhere CPU-Zeit die Tatsache, dass die 3D Graphikkarten sehr leistungsstark und auch die CPUs weiterentwickelt und billiger geworden sind. Abgesehen von den besseren Hardwarevoraussetzungen findet ein Mehreinsatz von künstlicher Intelligenz statt, weil es einen Wunsch des Kunden nach einer „realistischeren“ virtuellen Realität gibt.

Seit The Sims hat die Spieleindustrie begriffen, dass es ein Verlangen nach der *phänomenalen Gestalt* gibt. Es ist der Wunsch des Kunden so viel Realismus wie möglich in den Spielen zu haben. Somit ist die *phänomenale Gestalt*, ein Charakter,

[14] *The Sims*: Städtebau Simulation von Electronic Arts

[15] *roundtables:* Diskussionsrunden, in denen neueste Strategien und Techniken diskutiert werden

der sich selbst reflektiert, Hunger, Durst, ein Bedürfnis nach Liebe, Schlaf und Ähnlichem hat, unabdinglich.

Ein Grund für die weit verbesserte künstliche Intelligenz in Computerspielen, ist, laut Woodcock weiterhin, dass erstmalig gute AI Techniken gefunden und entwickelt wurden. Ein Beispiel hierfür ist die belief/ desire/ intention- Architektur, die Richard Evans für *Black and White*[16] umgesetzt hat, auf die im nächsten Kapitel eingegangen wird. Prinzipiell wandelt sich die AI von Charakteren vom geskripteten zu individuellem, realistischem, abgeleiteten Verhalten. Somit entwickelt sich die Charakter AI vom Reaktions- zum Entscheidungsverhalten, was dazu führt, dass die AI intelligenter und motivierter erscheint. Mit dieser Voraussetzung können natürlichere *NPCs*[17] entwickelt werden, die sich in einer merklich organischeren Welt bewegen.

Prinzipiell steht die Spiele AI noch am Anfang ihrer Entwicklung, aber ihr Einfluss führt dazu, dass sich die Spiele sehr stark verändern; weg vom geskripteten, monotonen Charakterverhalten, hin zur Entscheidungsfindung, um Charaktere mit einem autonomen Leben und intelligentem Verhalten zu erschaffen.

[16] *Black and White*: Götter-Simulation von Lionhead

[17] *NPC:* non playing character; Spielfiguren, die selbstständig agieren

Autor: René Greulich Matrikelnr.: 110 18781
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke Zweitprüfer: Prof. Dr. E. Ehses

3.0

AI in Bezug auf das Verhalten von Charakteren in Computerspielen

Charaktere

Charaktere oder *Actors* sind die Spielfiguren in einem Computerspiel. Klassische Beispiele sind Lara Croft, Mario, Duke Nukem und Indiana Jones.

Abbildung 02:
Charaktere

Man unterscheidet zwei Arten von Charakteren. Es gibt *Playing Characters (PCs)* und *Non Playing Characters (NPCs)*. Bei den PCs handelt es sich um Charaktere, die der Interaktion des Spielers folgen, d.h. die PCs sind die „eigentlichen Spielfiguren“, die sich steuern lassen. Der Benutzer schlüpft in die Rolle eines Playing Characters. Beim *Mensch- Ärgere – Dich- Nicht* ist, wenn man mit einer der Figuren einen Spielzug macht, diese Figur der PC. Lara Croft, Mario, Duke Nukem und Indiana Jones sind Playing Characters. Spielfiguren, die losgelöst von der Interaktion des Spielers agieren, sind NPCs. Auf ihr Verhalten hat man keinen direkten Einfluss, allerdings reagieren sie, wenn sie einer gewissen Intelligenz folgen, auf die

Handlungen des PCs. Verärgert man beispielsweise einen NPC, dann kommt es vor, dass er böse wird, einen gegebenenfalls anschreit oder vielleicht sogar angreift.

NPCs führen innerhalb eines Computerspiels ein autarkes Leben. Handelt es sich bei dem NPC z.B. um einen Bäcker, dann könnte dieser, ohne dass man als Spieler Einfluss auf ihn ausübt, um drei Uhr morgens aufstehen, frühstücken, sich waschen, zu seiner Bäckerei gehen, Brot backen und um sechs Uhr seinen Laden aufmachen. Dies wiederum gibt einem als Spieler die Möglichkeit mit ihm zu sprechen, wichtige Informationen zu erfahren oder einfach nur den virtuellen Hunger des PCs zu stillen. Diese kleine Szene verrät in Bezug auf Charaktere zwei Fakten. Zum einen glaubt der Spieler in der Handlung selbst zu agieren, obwohl er eigentlich nur seinen PC steuert, er spricht von ‚(s)ich' und seinen Handlungen. Hierbei ist es gleichgültig, ob man sich im *First Person View*[18] oder im *Third Person View*[19] befindet. Beim First Person View spielt man das Spiel in der Ich- Perspektive, d.h. man hat ein Blickfeld, als würde man selber in der Spielsituation stehen. Oftmals sieht man, z.B. bei Shootern nur ein Gewehr in einer Höhe, in der man selbst ein entsichertes Gewehr tragen würde. Beim Third Person View läuft der PC sozusagen ständig vor einem her. Der First Person View wird häufig bei sogenannten *Ego-Shootern* [20], der Third Person View bei sogenannten *Jump- and- Run- Spielen* [21]eingesetzt.

[18] *First Person View*: Ich- Perspektive

[19] *Third Person View*: hierbei schaut man permanent, während man sie steuert, auf seine Spielfigur

[20] *Ego- Shootern:* „Baller"- oder Kriegsspiel

[21] *Jump- and- Run- Spiele:* hierbei steuert man eine Spielfigur durch einen Parkur, bei dem man Hindernisse überwinden muss

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Abbildung 03: **First Person View**	**Abbildung 04:** **Third Person View**
	
sichtbarer Charakter: NPC	sichtbarer Charakter: PC

Zum anderen zeigt die eben dargestellte Spielszene, dass ein Charakter, in diesem Fall der Bäcker, gegebenenfalls situationsbedingt „eigenmächtig“ handeln kann. Dieses eigenmächtige Handeln ist Teil der Artificial Intelligence eines Charakters.

Prinzipiell lässt sich die AI in Spielen in die Gruppen *Lokomotion* (motor skills), *Steering* und *Motivation* (Action Steering) unterteilen. Die AI bestimmt das Verhalten bzw. Handeln eines NPCs in Bezug auf eine bestimmte Situation bzw. Herausforderung.

Lokomotion – Motor Skills[22]

Beim Lokomotion beschäftigt man sich mit der Frage, wie sich Spielfiguren auf graphischer Ebene in der künstlichen Welt bewegen. Mit der Bewegung sind motorische Fähigkeiten, wie z.B. das Gehen, Laufen, Sitzen, Reden, Schwimmen, etc. gemeint.

[22] *motor skills:* motorische Fähigkeiten

Prinzipiell besteht jede Animation eines Charakters im 2D Bereich aus einer Abfolge von einem bis mehreren Bildern, die frame[23] für frame, nacheinander abgespielt werden. Eine Standphase besteht beispielsweise aus einem, eine Pausenanimation aus fünf oder mehr Bildern.

Damit ein Charakter für alle Situationen in einem Spiel entsprechende Animationen zur Verfügung hat, bedarf es, ähnlich wie beim Trickfilm, einer Vielzahl von Einzelanimationen, die miteinander kombiniert werden müssen. Die Animationen selbst müssen mit einer gewissen Intelligenz gekoppelt sein. Besteigen beispielsweise zwei Charaktere einen Berg, wobei einer einen schweren Rucksack trägt, dann sollte das Gehen der Spielfigur mit dem Rucksack wesentlich schwerfälliger erscheinen.

Um allen Situationen befriedigend zu begegnen, muss entweder viel individueller source code implementiert werden oder man bedient sich der künstlichen Intelligenz und somit des Lokomotions. Mit Hilfe eines *Physikmanagers*[24], der in bestimmten Momenten selbst die benötigten Animationen des Charakters ausführt, organisiert der Charakter seine Animationsauswahl selbst. Hierbei bedarf es sowohl einer intelligenten Spielfigur, als auch einer intelligenten *Umgebung*[25], sowie intelligenter Utensilien. Zum einen muss der Charakter erkennen, auf was für einem Untergrund er steht. Diese Informationen liefert ihm die intelligente Umgebung. Im Falle eines Berges beinhaltet die Umgebung die Informationen:

- Areal/ Area: Berg
- Zustände: Momentane Steigung, Geröll, etc.

Anhand dieser Daten entscheidet die intelligente Spielfigur welche Animationen er zu welchem Zeitpunkt darstellen muss, aber z.B. auch wie schnell er vorangehen soll, um realistische Bewegungen zu simulieren. Dies hat zur Folge, dass eine ständige individuelle Programmierung der Animationsabläufe nicht mehr notwendig ist. Ändert

[23] *frame*: Bild (Bildfolge von Computerspielen wird in Bilder pro Sekunde (frames per second) bestimmt)

[24] *Physikmanager:* Fachbegriff ist physical controler

[25] *Umgebung*: Fachbegriff ist area

sich eine Komponente, z.B. die Steigung oder der Untergrund, reagiert der Charakter sofort mit der entsprechenden richtigen Animation. Genauso verhält es sich mit den intelligenten Utensilien. Der Rucksack, welchen der Charakter auf dem Rücken trägt, liefert der Spielfigur ebenfalls Informationen, wie z.B. das Gewicht, die es dem Charakter ermöglichen, realistische, intelligente Bewegung zu simulieren.

Das Lokomotion ist somit der Teil der künstlichen Intelligenz in Computerspielen, der die Bewegungen von Charakteren, mit Hilfe von intelligenten *Areas* und Utensilien, steuert.

Steering

Steuerung von Entitäten

Das Steering ist der Teil der Artificial Intelligence, der der Steuerung einer Entität in einer Spielszenerie dient. Prinzipiell wird hierbei eine Entität von einem Punkt A zu einem Punkt B manövriert, ohne die „Naturgesetze" der virtuellen Welt zu verletzen. Computerspiele versuchen die reale Welt bestmöglich abzubilden, somit sollte ein Charakter, wie ein Mensch, nicht durch Wände, Bäume, etc. gehen können, sondern von den Hindernissen aufgehalten werden.

Ein typisches Beispiel für das Steering ist die Verfolgungssituation. Die Spielfigur befindet sich entweder in der *Verfolgerrolle*[26] oder in der Rolle des *Verfolgten*[27]. Prinzipiell bedarf die Verfolgungssituation nur der momentanen Positionen des Verfolgers und des Verfolgten. Der Verfolger berechnet den kürzesten Weg zum Gejagten, während dieser den bestmöglichen Ausweg generiert. Schon hier wird einem bewusst, dass nur mit diesen beiden Informationen kein intelligentes Charakterverhalten entstehen kann. Beide Charaktere benötigen

[26] *Verfolgerrolle*: Fachbegriff ist chasing

[27] *Verfolgerrolle*: Fachbegriff ist evading

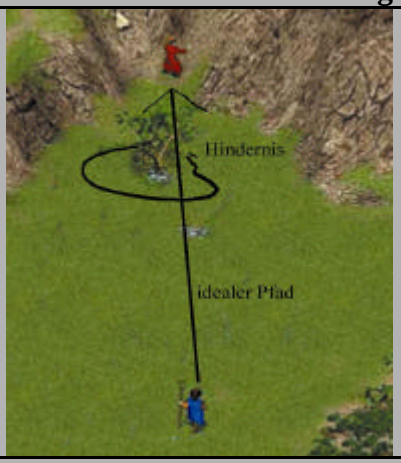
Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Umgebungsinformationen, wie z.B., ob sich der Charakter in einem Raum befindet. Es könnten sich folgende Fragen stellen: Hat dieser Raum, im Falle des Verfolgten, zwei oder mehr Ausgänge, kann der Verfolgte erahnen, durch welchen Eingang der Verfolger den Raum betreten wird ? Wie sieht der kürzeste Weg aus dem Raum aus, bei dem keine „Naturgesetze" verletzt werden? Kennt sich der Verfolger in diesem Gebäudekomplex aus? Vielleicht ist ein bestimmter Raum der Nächste für den Verfolger, hat aber keinen weiteren Ausgang, usw. Wie man erkennen kann, entwickelt sich ein scheinbar kleines Problem schnell zu einem äußerst komplexen.

Pathfinding

Ursprünglich entstand das Problem einer effektiven und logischen Pfadfindung in der Robotik. Hier benutzte man anfangs *potential functions*, um exakte Pfade zu berechnen. Die Basis der potential functions ist der direkte Vektor zwischen zwei Objekten, mit dessen Hilfe der kürzeste Weg ermittelt werden kann. Das Problem ist, dass sich in einer *Lokation*[28] meist Hindernisse befinden, sodass selten ein direkter Weg von A nach B existiert und ein Ausweichmanöver notwendig ist.

Abbildung 05:
Ausweichmanöver notwendig



Eine zweite Möglichkeit Pfade zu berechnen sind *Objectmaps*. Hierbei handelt es sich um vorgefertigte Pläne, auf denen alle Hindernisse und die Möglichkeit sie zu

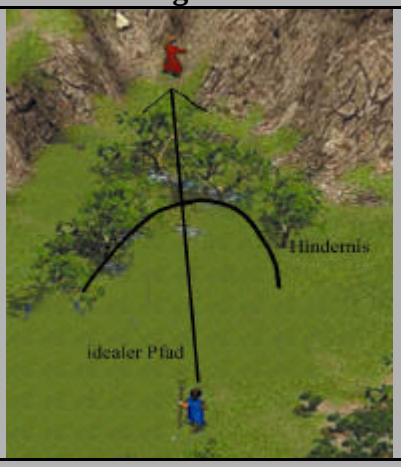
[28] *Lokation*: hierbei handelt es sich um den (sichtbaren) Bereich/ das Ge lände/ den Ort des Geschehens, indem sich der PC befindet

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

umgehen, festgelegt sind. Im Gegensatz zu den potential functions können hiermit keine beweglichen Hindernisse berücksichtigt werden.

Ein großer Nachteil der potential functions ist, dass kein Pfad generiert werden kann, wenn Hindernisse in U-Form vorhanden sind, weil die Suche dann viel zu lange dauert.

Abbildung 06:
Aufgrund der U-Form versagt der Algorithmus



Aufgrund der Tatsache, dass diese beiden Pfadfindungsmechanismen viel zu ineffizient und ineffektiv sind, werden sie in der Spieleprogrammierung nicht verwendet.

Gutes Pathfinding

Eine schnelle, gute, effektive und logische Pfadverfolgung ist eine erweiterte Intelligenz eines Charakters und trägt dazu bei, die Qualität eines Computerspiels zu erhöhen.

Das Pathfinding muss schnell sein, damit die Performanz des restlichen Spiels nicht unter den Algorithmen leidet. Das Pathfinding muss gut und effektiv sein, damit nur Pfadkoordinaten ermittelt werden, die nicht die Naturgesetze des Spiels verletzen, um zu vermeiden, dass die Bewegung der Spielfigur irreal erscheint. Das Pathfinding

muss logisch sein, um das *Gameplay*[29] nicht zu verletzen. Manchmal ist der kürzeste Weg nicht der Logischste. Halten sich beispielsweise zwei NPCs in der Nähe eines Berges auf, wobei es sich bei dem einen um einen Händler, der gerade mit seiner Kutsche vom Markt kommt, bei dem anderen um einen leidenschaftlichen Wanderer handelt, dann können sich ihre generierten Wege durchaus unterscheiden. Berechnet man für beide NPCs vom gleichen Startpunkt den Weg zum gleichen Ziel, dann nimmt der Wanderer die Wanderroute über den Berg und der Händler folgt der gesicherten Straße. Beide handeln nach ihrer persönlichen „künstlichen" Intelligenz.

Pathfinding und die Graphentheorie

Prinzipiell werden die (kürzesten) Pfade mit Methoden ermittelt, die auf der *Graphentheorie* basieren. Hierbei handelt es sich im Speziellen um den A^*[30]- und den *Dijkstra*- Algorithmus. Welcher dieser beiden Algorithmen in der Spieleprogrammierung benutzt werden sollte, wird noch diskutiert werden.

Motivation – Action Steering

Entscheidungsfindung

Der dritte Bereich der Artificial Intelligence ist die Motivation der Charaktere. Hierbei handelt es sich um Mechanismen der *Entscheidungsfindung*, d.h. um die Reaktion des Charakters auf Vorkommnisse in seiner virtuellen Welt. Charaktere müssen, damit ein Spiel interessant ist, auf weltliche Herausforderungen intelligent reagieren. Wenn ein NPC hungrig ist, dann muss er sich Nahrung verschaffen, wenn er häufig beleidigt wird, dann greift er gegebenenfalls einen anderen Charakter an. Stellt er plötzlich fest, dass er den Kampf verlieren wird, dann tritt er den Rückzug an.

[29] *Gameplay*: Spielablauf, Spielaufbau, Spielführung

[30] *A**: gesprochen – A Star

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Vorlagenbasierte KI

Der einfachste Algorithmus einer KI- Anwendung basiert auf sogenannten Pattern, bei denen anhand von vordefinierten Regeln Reaktionen von Entitäten ausgelöst werden. Das klassische Beispiel für eine solche Patter-KI ist das bekannte Spiel *Space Invaders*[31]. Bei diesem Spiel gibt es einen Angriffs- Pattern, dem die feindlichen Raumschiffe folgen, um den Spieler abzuschießen. Dieser Angriffs-Pattern besteht aus Koordinaten, die zusammen die Flugroute eines feindlichen Raumschiffes beschreiben. Diese Koordinaten werden anhand der Positionskoordinaten des Raumschiffs des Spielers angepasst, damit es die feindlichen Raumschiffe immer schaffen, den Spieler zu treffen. Alle Raumschiffe eines Typs verfolgen den gleichen Angriffs- Pattern. Aufgrund der Einfachheit ist man als Spieler schnell in der Lage, auf die Manöver der feindlichen Angreifer zu reagieren. Obwohl das Verhalten der Verfolger sehr primitiv ist, handeln die Raumschiffe dennoch nach einer gewissen Intelligenz. Um der Einfachheit entgegenzuwirken werden die Koordinaten anhand eines Zufallsgenerators verifiziert, sind aber dennoch nach kurzer Zeit berechenbar.

Aufgrund der Tatsache, dass das Gameplay eher schlicht ist, kann die Pattern-KI durch eine sogenannte Skript- KI erweitert werden. Die Skript- KI spezialisiert die vorlagenbasierte KI indem, abhängig vom Spielstand, verschiedene Pattern ausgeführt werden, mit dem Ziel, dass sich der Spieler öfter auf neue Situationen einstellen muss.

Aus heutiger Sicht erscheint die vorlagenbasierte KI sehr primitiv. Zu der Zeit von Space Invader war dies allerdings revolutionär und Space Invaders eines der beliebtesten Spiele.

Entscheidungsfindung

Bei einem Spiel wie Space Invaders findet keine Entscheidungsfindung statt. Die feindlichen Raumschiffe greifen ständig in Form des gleichen Patterns an. Sie lernen nicht aus ihren Erfolgen oder Fehlern. Bei dem Spiel *Black und White*, welches

[31] *Space Invaders*: Kriegsspiel mit Raumschiffen

neben The Sim als hervorragendes Beispiel für die künstliche Intelligenz gilt, ist das anders. Hier reagieren die Charaktere nicht nur auf ihre Umwelt, hier lernen sie sogar aus ihren und den Entscheidungen anderer.

Black and White ist das erste Spiel, in dem eine grandiose realistische Entscheidungsfindung stattfindet. Wenn man bedenkt, dass zwischen Black and White und Space Invaders mindestens zwanzig Jahre liegen, ist es sehr erstaunlich, dass es erst jetzt ein Spiel mit derart intelligenten NPCs gibt.

Vor Black and White war die Entscheidungsfindung in Computerspielen prinzipiell bis ins letzte Detail programmiert. Ein Lerneffekt fand im eigentlichen Sinne nie statt.

Richard Evans, Chefprogrammierer der Artificial Intelligence bei Lionhead, weist darauf hin, dass das *Spieledesign* [32]von Black and White nur mit einer dynamischen Entscheidungsfindung funktioniert. In Black and White geht es im Groben darum, als Gott Kreaturen zu seinem Gefolge zu machen, um die Welt zu beherrschen. Die Kreaturen müssen zwei Anforderungen erfüllen. Zum einen sollen sie dem Spieler das Gefühl geben, sich tatsächlich mit einer realen Figur zu beschäftigen. Aus diesem Grund sollen die Figuren *plausibel*, *anpassungsfähig* und *liebenswert* erscheinen. Des weiteren stellen die Figuren Helfer für den Spieler dar, die entsprechend der verfolgten Kampagne die Welt zu beherrschen, trainiert werden sollen. Evans weist darauf hin, dass die Herausforderung darin bestand, die gewollte Autonomie der Charaktere und deren Unterwürfigkeit dem Spieler als Gott gegenüber, zu vereinen.

[32] *Spieldesign*: Fachbegriff ist game- design; Spielaufbau, Spielstrategie

Damit die Charaktere wie eine Person wirken, müssen sie, wie erwähnt, plausibel, anpassungsfähig und liebenswert sein. Um dies zu ereichen bedarf es eines *psychologischen Plausibilitätsagenten* [33] [34] (PPA).

[33] *Agent:* Ein Agent ist eine physische oder virtuelle Entität,

- die selbstständig in einer Umwelt agieren kann,
- die direkt mit anderen Agenten kommunizieren kann,
- die durch eine Menge von Absichten angetrieben wird (in Form von individuellen Zielen, Befriedigungs- und Überlebensfunktionen, die sie versucht zu optimieren),
- die eigene Ressourcen besitzt,
- die fähig ist, ihre Umwelt wahrzunehmen (allerdings nur in bestimmtem Ausmaß),
- die nur eine partielle Repräsentation ihrer Umwelt besitzt,
- die bestimmte Fähigkeiten besitzt und Dienste offerieren kann,
- die sich ggf. selbst repräsentieren kann,
- deren Verhalten darauf ausgerichtet ist, ihre Ziele, unter Berücksichtigung der ihr zur Verfügung stehenden Ressourcen und Fähigkeiten, zu befriedigen und die dabei auf ihre Wahrnehmung, ihre internen Modelle und ihrer Kommunikation mit anderen Agenten (oder den Menschen) angewiesen ist.

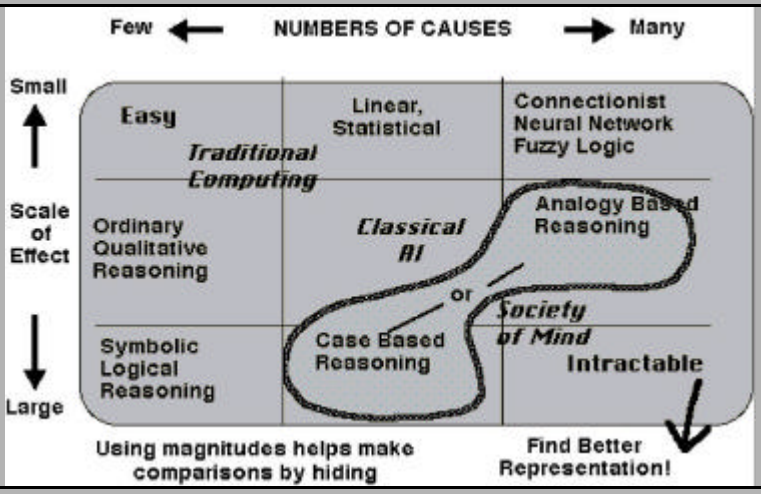
[34] *psychologischen Plausibilitätsagenten :* Fachbegriff ist psychologically plausible agent

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

belief/ desire/ intention- architecture

Der PPA basiert auf der *Wertevorstellungen / Verlangen/ Intension - Architektur*[35]. Das Entscheidende dieser Architektur ist, dass nicht, wie normalerweise üblich, nur eine Form der Schlussfolgerung für die Entscheidungsfindung zur Verfügung steht, wie z.B. nur *Fuzzy Logic* [36] oder nur neuronale Netzwerke, sondern eine Vielzahl von Schlussfolgerungsmethoden. Damit dies möglich ist, basiert die Architektur auf der *Theoriematrix*[37] von Marvin L. Minsky.

Abbildung 07:
Theoriematrix



Zusammenhang zwischen Ursache & Wirkung

Bei der Theoriematrix geht es um die Frage, wie Wissen abhängig von Ursache und Wirkung abgeleitet werden kann und welche Art von Logik in welchen Situationen verwendet werden soll. Wenn die Ursache und die Wirkung sehr gering sind, dann handelt es sich um ein triviales Problem, zu dessen Lösung eine traditionelle computerunterstützte Auswertung der Situation ausreicht. Haben allerdings viele Ursachen eines Problems nur eine geringe Wirkung, dann sollte Fuzzy Logic oder Neuronale Netzwerke verwendet werden, usw.

---

[35] *Wertevorstellungen / Verlangen/ Intension Architektur*: Fachbegriff ist belief/ desire/intention architecture

[36] *Fuzzy Logic:* Bei der Fuzzy Logic besteht die Wertemenge nicht nur aus wahr und falsch oder 1 und 0, d.h. nicht nur aus „ganzwertigen“ Beurteilungen , sondern aus vielen Einzelfacetten, wie z.B. von 1.001 bis 3.825, um sehr feine Logikabstufungen beurteilen zu können.

[37] *Theoriematrix:* Fachbegriff ist theory matrix

Für Black and White bedeutete dies, dass die Wertevorstellung bzw. die Einstellung einzelner Objekte als eine Liste von Attributen dargestellt und deren Logik anhand von *Junktoren*[38]- und *Quantoren*[39] in Bezug auf die Welt hergeleitet wurde. Die Wertevorstellungen für einen Typ oder eine Gruppe von Objekten wurde anhand von *Entscheidungsbäumen*[40] generiert und das Verlangen als *Perzeptron* [41] repräsentiert. Die Wertevorstellungen fallen somit mehr in die Aussagen- und das Verlangen in die Fuzzy Logic.

---

[38] *Junktoren:* logische Symbole, wie z.B. ? (und) , ? (oder), ? (non), die in der Aussagenlogik zur Bildung von (zulässigen) Formeln (z.B. A ? B) verwendet werden. Der Wahrheitswert solcher Formeln wird nur aufgrund der Wahrheitswerte von A und B definiert. Junktoren nennen sich auch Funktoren. ? ist einstellig, da hinter dem Negator nur eine Aussage stehen darf; ? und ? sind zweistellig, weil sie von zwei Aussagen umgeben sind.

[39] *Quantoren*: hierbei geht es um die aus dem aristotelischen Syllogismus bekannten All- und Existenzquantoren, die bisweilen auch Operatoren heißen.
Beispiel:

Allquantor:
Symbol: (? x)
Aussage: Alle Dinge sind
Beispiel: (Alle Dinge sind) rund (? x) Rx

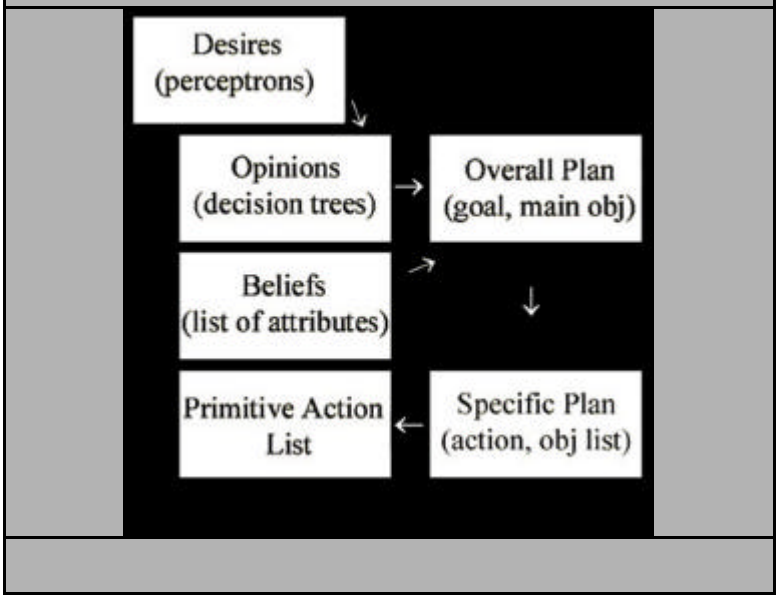
Existenzquantor:
Symbol: (? x)
Aussage: Etwas ist
Beispiel: (Etwas ist) rund (? x)Rx

[40] *Entscheidungsbäumen*: Fachbegriff ist decision trees

[41] *Perzeptron:* Modell eines lernenden Automaten zur Nachbildung von Wahrnehmungs- und Lernprozessen. Das Prozeptron benutzt Verfahren nach dem Prinzip der bedingten Wahrscheinlichkeit (Brockhaus). Das Perzeptron war eins der ersten Modelle von Neuronalen Netzwerken von McCullogh und Pitts(1943). Es diente dazu, die menschliche Schlussfolgerung aufzuzeigen. Es hat zwei Eingänge und einen Ausgang

Um einen *Handlungsplan* zu entwickeln, verfolgen/ be(tr)achten/ interpretieren die Kreaturen ihr Verlangen, ihre Wertevorstellungen und die prinzipiellen Möglichkeiten, die sie aufgrund der Handlung und ihren Fähigkeiten haben.

Abbildung 08:
belief/ desire/intention- architecture



Der entstandene Plan wird daraufhin in einzelne Komponenten unterteilt, um spezifisch vorgehen zu können. Somit entsteht eine Liste von Aktionen, die, wenn die Kreatur sie abgearbeitet hat, zu einer abgeschlossenen Handlung führt. Eine Kreatur könnte z.B. den Plan fassen, eine Stadt anzugreifen. In diesem Moment plant es erst einmal, ein einzelnes Haus oder eine einzelne Person anzugreifen. Diese Aktion wird nun in einzelne *Tasks* [42] unterteilt, die nach der Planung nacheinander ausgeführt werden.

Black and White bildet die Realität dermaßen gut ab, dass es sogar das Prinzip der *erkenntnistheoretischen Wahrscheinlichkeit*[43] verfolgt. Dies bedeutet, dass es in Black and White immer einen Grund geben muss, weshalb eine Kreatur in einem bestimmten Zustand ist, denn eine Kreatur erweitert sein Wissen, wie Babies, über „Sinneseindrücke". Die Kreaturen besitzen somit auch nur das Wissen, welches sie im Laufe des Spiels erfahren haben und können somit nur in den mentalen Zuständen

[42] *Task:* Teilaktion

[43] *erkenntnistheoretischen Wahrscheinlichkeit:* Fachbegriff ist epistemic versimilitude

sein, die auf ihrem Wissen aufbauen. Für ein Computerspiel ist die Umsetzung der erkenntnistheoretischen Wahrscheinlichkeit phänomenal. Die Kreaturen sind natürlich erst dadurch handlungs- und entscheidungsfähig, weil sie ein gewisses Grundwissen besitzen.

Das Verlangen ist ähnlich umgesetzt. Die Kreaturen haben nicht einfach irgendein Verlangen, sondern es muss immer einen Grund für ihr Verlangen geben. Ein Charakter ist nicht grundlos wütend. Wahrscheinlich hat er eher beobachtet, wie der Spieler etwas zerstört hat und möchte das Verhalten des „Erwachsenen“ kopieren oder vielleicht wurde er auf irgendeine Art und Weise verletzt und weiß sich nicht anders auszudrücken.

Die Möglichkeit verschiedene Reize als Auslöser für ein Verlangen und damit für die adäquate Handlung zu haben, verleiht dem Charakter nicht nur eine künstliche Intelligenz, die ihm das Verhalten erst ermöglicht, sondern auch realistische, individuelle Charakterzüge, die es erlauben, verschiedene Individuen von Kreaturen oder NPCs zu entwickeln. Somit kann es Charaktere geben, die nur ein Verlangen nach Nahrung verspüren, wenn sie kurz vor dem Verhungern sind. Andere verspüren gegebenenfalls einen gewissen Hungerneid und müssen etwas essen, weil sie gerade jemanden beim Essen beobachten. Wieder andere essen, weil sie gerade traurig sind. Manche haben mehrere dieser Eigenschaften. Mit dieser Vorgehensweise ist der Moment der Spieleprogrammierung erreicht, indem ein Heer von Intelligenz- und Verhaltensklone von individuellen „Lebewesen“ abgelöst wird.

Verformbare Agenten

Wie man feststellen kann, zielt die AI – Architektur von Black and White auf die Fähigkeit intelligenter Charaktere zu lernen. Die Aufgabe der Figuren ist ihr Wissen zu erweitern, wobei die Spielsituationen so aufgebaut sind, dass es eine Möglichkeit zum erweiterten Lernen gibt. Um Entscheidungen treffen zu können, müssen die Kreaturen lernen, welche Methoden sie in bestimmten Situationen anwenden müssen, welche Objekte wie, wann und wo benutzt bzw. angewandt werden

und wie und wann ihr Verlangen eintritt. Des weiteren müssen sie herausfinden, wie man überhaupt etwas macht und was etwas ist.

Das Lernen der Kreaturen wird durch verschiedene Mechanismen ausgelöst. Prinzipiell geschieht dies durch das Feedback des Spielers, indem dieser sie für bestimmte Handlungen belohnt oder bestraft. Des weiteren erweitert die Kreatur ihr Wissen durch Befehle, die ihr gegeben werden. Aufgrund der Tatsache, dass die Kreaturen sehr aufmerksam sind, lernen sie sogar durch die Beobachtung des Verhaltens und der Handlungen anderer Personen, wie z.B. des Spielers. Einer der wichtigsten Punkte ist jedoch die Reflektion der eigenen Erfahrungen. Das Lernen basiert so gesehen auf sogenannten *learning opinions*[44].

[44] *learning opinions:* Lernen durch Erfahrung (Learning opinions basieren auf dynamischen Entscheidungsbäumen)

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Liebenswürdige Agenten

Wie man so langsam erahnen kann ist Black and White momentan das beste Beispiel für die fortschrittlichste künstliche Intelligenz in Computerspielen. In Bezug auf die sogenannten liebenswürdigen Agenten wurde eine noch sehr interessante Intelligenz implementiert. Evans war der Meinung, dass sich ein Spieler dann zu einer Kreatur hingezogen fühlt, wenn sich diese zum Spieler hingezogen fühlt. Aus diesem Grund gab man den Kreaturen die Fähigkeit, ein mentales Profil des Spielers zu erstellen, indem die Kreaturen versuchen, die Handlungen des Spielers zu beobachten, zu analysieren und zu verstehen. Der Grund dafür ist, dass die Kreaturen Entscheidungen im Sinne des Spielers treffen sollen. Laut Evans sind Agenten in Computerspielen so etwas wie *Autisten*. Sie können das Verhalten von Objekten, NPCs und dem Spieler wahrnehmen und analysieren, aber sie können kein Modell für deren Verhalten erstellen, um Handlungen der Objekte für die Zukunft vorherzusagen. Bei Black and White findet immerhin eine sehr genaue Analyse des Spielers statt, mit dem Ziel seinem Meister, immerhin ist der Spieler ein Gott, zu helfen und mit ihm zu spielen. Der Höhepunkt der künstlichen Intelligenz der Kreaturen ist allerdings, da eine allzumenschliche Intension verfolgt wird, ein „Buhlen“ um die Aufmerksamkeit des Meisters.

Prinzipiell steht fest, dass den Kreaturen ein gewisses Wissensspektrum gegeben werden muss, da sie sonst keine Entscheidungen treffen werden und in ihrer eigenen Welt verloren sind.

Fazit

Black and White ist ein sehr schönes Beispiel, wie die künstliche Intelligenz das Charakterverhalten in Computerspielen sehr interessant gestalten kann und was für unglaubliche Szenarien dadurch möglich sind. In Black and White kann tatsächlich von einer Motivation im eigentlichen Sinne gesprochen werden. Die Wertevorstellungen / Verlangen/ Intension- Architektur bietet die Möglichkeit, einen Charakter gleichzeitig autonom handeln zu lassen, aber auch hinsichtlich bestimmter Handlungen und Ziele zu trainieren. Natürlich handeln die Charaktere weniger autonom, je mehr sie trainiert werden. Heutzutage besteht noch das Problem, dass die

Ziele und die Möglichkeiten diese Ziele zu erreichen endlich sind, d.h. es gibt eine festgelegte Anzahl von Situationen bzw. zu lösende Aufgaben, die die Kreaturen meistern müssen, und dass die Kreaturen ihre Aufgaben immer lösen. Evans ist allerdings der Meinung, dass bald Charaktere entwickelt werden können, deren Ziele unendlich sein werden. Natürlich greifen die Charaktere zur Lösung ihrer Probleme auf eine *Planungsbibliothek* zurück. Eines Tages, so Evans, bedarf es dieser Bibliothek nicht mehr und die Planung geht dann tatsächlich nur von der Kreatur aus. Der Vorteil wäre, was die Charaktere noch menschlicher erscheinen lassen würde, dass die Charaktere auch einmal versagen, d.h. dass sie eine Situation einmal nicht meistern können. Weiterhin wird es eines Tages möglich sein, äußerst viele Modelle von Charakteren zu verwirklichen, was zu noch realistischeren Welten führen wird.

Interessant ist, dass bei Black and White viele grundsätzliche Fragen mit Hilfe der Philosophie beantwortet wurden, was mit der philosophischen Ausbildung Evans zusammenhängt. Ein Philosoph, auf dessen Lehre Teile von Black and White aufbauen, ist z.B. Paul Grice.

Autor: René Greulich Matrikelnr.: 110 18781
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke Zweitprüfer: Prof. Dr. E. Ehses

4.0

Steering

Pathfinding- Algorithmen

Einleitung

Das Steering ist, wie schon erwähnt, der Teil der Artificial Intelligence von Computerspielen, der der Steuerung einer Entität in einer Spielszenerie dient. Prinzipiell gibt es eine Vielzahl von Möglichkeiten den Pfad einer Entität zu berechnen, allerdings ist das Grundproblem immer dasselbe. Das Problem ist, so banal es klingt, Hindernisse zu umgehen, denn nur, wenn sich ein Objekt realistisch in seiner Welt bewegt, entsteht die Illusion eines naturgetreuen Verhaltens.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Algorithmen, basierend auf der Ignorierung des Hindernisses

Die einfachste Methode das Pathfindingproblem zu lösen ist, die Hindernisse so lange zu ignorieren, bis eine Kollision stattfindet.

Pseudocode: Algorithmus der Ignorierung eines Hindernis

```
Wähle eine Richtung, in der du dich auf das Ziel zu bewegst
while(noch nicht am Ziel)
{
    if(Weg frei)
        Bewege Dich in diese Richtung
    else
        Wählen eine neue Richtung , anhand einer Hindernisumgehungsstrategie
}
```

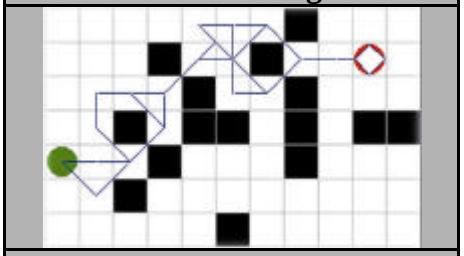
Diese Methode bedarf weniger Informationen. Man muss nur die relative Position der zu bewegenden Entität kennen, die Position des Ziels und der Blockade in der unmittelbaren Umgebung. Prinzipiell könnte man diese Methode für viele Spielsituationen verwenden.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Hindernisumgehungsstrategien

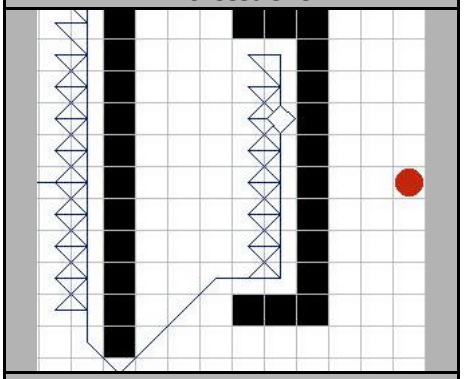
Solange die Hindernisse klein und konvex sind besteht die Möglichkeit, die Entität, wenn sie mit einem Objekt kollidiert, per Zufallsmechanismus ein wenig zur Seite zu bewegen und anhand der neuen Position die Berechnung fortzuführen.

Abbildung 09: Hindernisumgehung bei kleinen Hindernissen durch Verschiebung



Sind die Hindernisse allerdings groß oder konkav, kann es passieren, dass die Entität keinen Weg findet, oder dass sie sehr lange braucht, um den Weg zu berechnen.

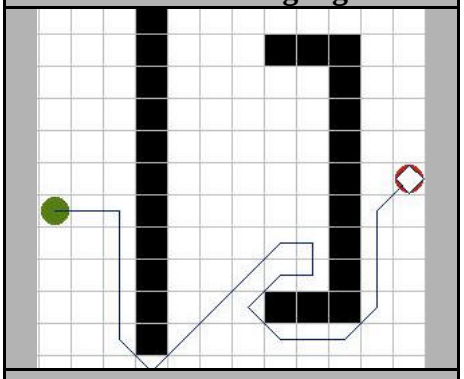
Abbildung 10: Endlossuche



Eine Möglichkeit dieses Problem zu umgehen, ist eine Welt zu erschaffen, in der es nur kleine, konvexe Hindernisse gibt.

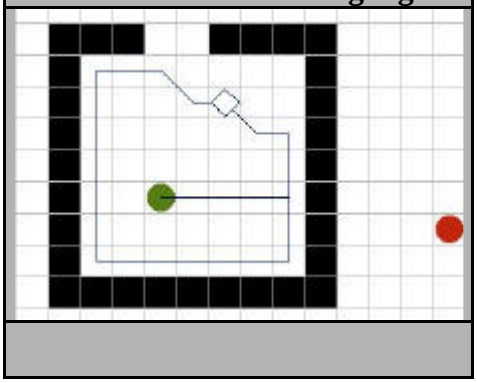
Ist ein Hindernis sehr groß, dann kann man einen Weg nach der „*Vortastenmethode*[45]“ generieren, wobei der Weg anhand der Umrisse der Hindernisse berechnet wird. Bei dieser Methode muss allerdings eine gutes Konzept entwickelt werden, um zu bestimmen, wann die Suche (nach der neuen Richtung) beendet ist. In der Regel hat man die Richtung gefunden, wenn man bei der *Umrissverfolgung* eine Richtung erhält, in der man sich zu Beginn der Umrisssuche bewegen wollte.

Abbildung 11:
Umrissverfolgung



Auch diese Methode funktioniert in vielen Fällen; sie kann allerdings dazu führen, dass sich die Entität permanent im Kreis bewegt, ohne jemals das Ziel zu erreichen.

Abbildung 12:
Endlose Umrissverfolgung



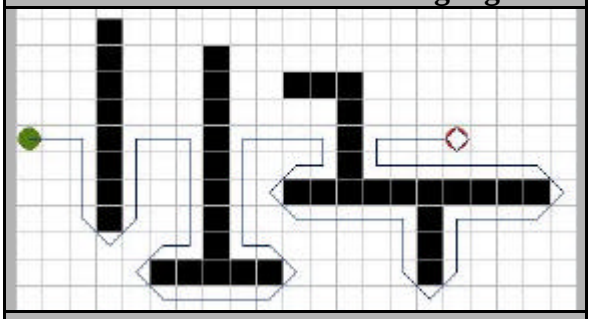
Um zu vermeiden, dass die Entität nicht zu ihrem Ziel findet, lässt sich die Umrissverfolgung hinsichtlich dieser Problematik verbessern. Hierzu verwendet man

[45] Vortastenmethode: Fachbegriff ist Umrissverfolgung

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

eine Methode, die der Robotik entliehen ist und der Fortbewegung, in diesem Fall von mobilen Robotern, dient. Man berechne den direkten Weg zwischen der momentanen Position P1 der Entität und dem Zielpunkt, der als virtuelle Suchgrenze dient. Zu Beginn sucht man auf der einen Seite dieser Linie. Sollte auf dieser Seite kein Weg zum Ziel führen, dann wird die virtuelle Grenze überschritten und auf dieser Seite weitergesucht. Wird diese nochmals überschritten, dann kann davon ausgegangen werden, dass man sich im Kreis bewegt hat. Die Suche wird abgebrochen, wenn der Punkt P1 wieder erreicht wird. Mit Hilfe dieser Methode wird definitiv ein Weg von A nach B berechnet, allerdings ist diese Route höchstwahrscheinlich nicht besonders ästhetisch und die Suche nicht sonderlich effizient.

Abbildung 13:
kontrollierte Umrissverfolgung



Das beste Ergebnis ergibt sich bei der Umrissverfolgung, wenn beide Techniken kombiniert werden. Dies bedeutet, dass solange die erste Methode verwendet wird, bis sich herausstellt, dass man sich im Kreis bewegt hat; danach wird auf die robustere Variante zurückgegriffen, um definitiv einen Weg zu berechnen.

Aufgrund der Tatsache, dass in heutigen Spielen die Grundproblematik nicht mehr die Hindernisse an sich sind, sondern dass es vielmehr einen Anspruch an Realismus gibt, werden diese einfachen Methoden, obwohl sie in den meisten Situationen funktionieren, nicht mehr verwendet.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Algorithmen, auf Graphen basierend

Heutige Pathfindingmethoden zur Berechnung des kürzesten Weges zwischen zwei Punkten A und B basieren in der Regel auf Graphen. Dies bedeutet, dass, im Gegensatz zu anderen Techniken, der gesamte Weg unter Berücksichtigung aller nicht beweglichen Objekte (vor)berechnet wird.

Definition des Graphen

„Wie bei vielen anderen der von uns untersuchten Problemkreise wurde auch bei Graphen erst in jüngster Zeit damit begonnen, sie von einem algorithmischen Standpunkt aus zu betrachten. Auch wenn einige der grundlegenden Algorithmen sehr alt sind, sind viele interessante Algorithmen erst im Laufe der letzten zehn Jahre entdeckt worden. Selbst triviale Algorithmen für Graphen führen zu interessanten Programmen, und die nichttrivialen Algorithmen, die wir kennen lernen werden, gehören zu den elegantesten und meist interessanten (wenn auch schwer verständlichen) bekannten Algorithmen“. [46]

Robert Sedgewick

Ein Graph besteht aus *Knoten* und *Kanten*. Bei den Knoten handelt es sich um Objekte. Eine Kante ist die Verbindung zwischen zwei Knoten. Ein *Weg* oder *Pfad* ist eine Anreihung von durch Kanten miteinander verbundener Knoten innerhalb eines Graphen. Bei einem *einfachen Pfad* kommt jeder Knoten nur einmal vor. Das Ergebnis der Suche nach dem kürzesten Weg ist ein einfacher Pfad. Bei einem *Zyklus* ist der erste und letzte Knoten identisch. Ein Graph ohne Zyklen ist ein *Baum*. Hierbei kann jeder Knoten über genau einen Weg von der Wurzel aus erreicht werden. Die Länge eines Weges ist gleich der Summe der Kantenlängen. Jeder Knoten außer der Wurzel hat einen *Elternknoten*, der eine Ebene über diesem *Kindknoten* liegt. Die Knoten einer Ebene nennt man *Schwesternknoten*. Ein Knoten, der keine Kindknoten besitzt, nennt man *Blatt*. Die Kanten können auch als *Äste* bezeichnet werden.

[46] Zitat: Sedgewick, Algorithmen in C++, S.474

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Breadth first search – Breitensuche

Bei der ebenenweisen Traversierung von Bäumen beginnt die Suche beim Startknoten (Wurzel) und überprüft alle direkten Nachbarknoten (Schwesternknoten), danach alle Knoten, die zwei, danach die drei Schritte bzw. Ebenen entfernt sind, usw., bis der Zielknoten gefunden wird. Prinzipiell bedeutet dies, dass die Knoten von links nach rechts und von oben nach unten traversiert werden. Alle nicht untersuchten Nachbarknoten eines Knotens werden in einer sogenannten Open- Liste gespeichert. Bei der Breitensuche handelt es sich in der Regel um eine *FIFO*[47] - Queue.

Pseudocode: Breitensuche

```
Queue Open
BreadthFirstSearch
    Node n, nn, s
    s.parent = NULL // Startknoten
    push s on Open
    while(Open ist nicht leer)
    {
        pop Knoten n von Open
        if(n ist Zielknoten)
            Konstruierte Pfad und beende Suche
        for jeden Nachfolger nn von n
        {
            if(nn wurde schon besucht)
                continue
            nn.parent = n
            push nn nach Open
        }
    }
```

[47] *FIFO*: first in first out

Die Breitensuche findet den kürzesten Weg um ein Hindernis, wenn alle Schritte die gleiche Länge haben. Das Problem ist, dass dies in der Regel nicht der Fall ist. Es sollte zumindest einen Unterschied zwischen der diagonalen und orthogonalen Schrittlänge geben. Ein weiteres Problem der Breitensuche ist, dass in alle Richtungen gesucht wird, anstatt die Suche in Richtung des Ziels zu begrenzen.

Bidirektionale Breitensuche

Im Gegensatz zur einfachen Breitensuche werden simultan zwei Breitensuchen, eine beginnend beim Start-, eine beim Zielpunkt, durchgeführt. Die Berechnung endet, wenn der Knoten der einen Suche, der Nachbarknoten der anderen ist. Die bidirektionale Breitensuche verringert den Suchaufwand erheblich. Allerdings ist auch diese Suchtechnik nicht effizient, weil, wie bei der normalen Breitensuche, die Suchrichtung nicht beachtet wird.

Dijkstra- Algorithmus

Im Gegensatz zur Breitensuche ist es mit Hilfe des Dijkstra- Algorithmus möglich, einen Graphen mit Kanten verschiedener Gewichtung zu durchsuchen. Bei jedem Schritt wird hierbei, der nicht kontrollierte Knoten, der sich am nächsten zum Startknoten befindet, betrachtet. Die aufsummierten Distanzen dieses Arbeitsknotens zum Startpunkt werden mit den Distanzen seiner Nachbarknoten verglichen. Der Knoten der kürzesten Distanz wird zur weiteren Untersuchung verwendet. Im Gegensatz zur Breitensuche, bei der eine FIFO- Queue verwendet wird, benutzt man beim Dijkstra- Algorithmus eine *priority queue*[48], bei der die Knoten in Reihenfolge der Distanzen zum Startknoten sortiert sind. Der Dijkstra basiert auf der Graphentheorie, auf die im Speziellen noch eingegangen wird.

[48] p*riority queue:* Prioritätsschlange

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Pseudocode: Dijkstra- Algorithmus

```
priority queue Open
DijkstraSearch
        Node n, nn, s
        s.cost = 0
        s.parent = NULL // Startknoten
        push s on Open
        while(Open ist nicht leer)
        {
                pop Knoten n von Open // und zwar den mit den geringsten Kosten
                if(n ist Zielknoten)
                        Konstruiere den Pfad und beende die Suche
                for jeden Nachfolger nn von n
                {
                        NeueKosten = n.cost + cost(n, nn)
                        if(nn befindet sich in Open und nn.cost <= NeueKosten)
                                continue
                        nn.parent = n
                        nn.cost = NeueKosten
                        if(nn ist immer noch nicht inOpen)
                                push nn nach Open
                }
        }
```

Der Nachteil des Dijkstra- Algorithmus ist, dass, wie bei der Breitensuche, die Suchrichtung (in Richtung Ziel) nicht beachtet wird.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Tiefensuche

Dieser Suchalgorithmus ähnelt der Breitensuche. Der Unterschied besteht darin, dass bei der Tiefensuche nicht ebenenweise, sondern tiefenmäßig vorgegangen wird. Eine tiefenmäßige Vorgehensweise bedeutet, dass nicht erst alle Schwesternknoten eines bestimmten Knotens besucht werden, sondern zuerst alle Nachkommen. Prinzipiell bedeutet dies, dass erst die Wurzel, dann der linke Unterbaum, dann der rechte Unterbaum besucht wird.

Im Gegensatz zur Breitensuche wird an Stelle einer FIFO eine *LIFO*[49] Queue (oder *Stapel* [*stack*]) verwendet, allerdings ist es vorteilhaft, bei der Tiefensuche eine rekursive Traversierung durchzuführen. Zuerst wird die Wurzel besucht. Da nicht beide Unterbäume eines binären Baums gleichzeitig besucht werden können, speichert man den rechten Unterbaum in einem Stapel und durchsucht den gesamten linken Unterbaum; danach den rechten.

Die Tiefensuche tendiert dazu, viel Zeit für einen äußerst ineffektiven Weg zu verschwenden. Um dies zu vermeiden, bedarf jeder Knoten der Information über die Länge des kürzesten Pfades, der bis zu diesem Punkt gefunden wurde. Dies verhindert, dass der Knoten noch einmal besucht wird, es sei denn, es gibt entweder einen kürzeren Weg oder einen genau so kurzen in einer tieferen Ebene. Zur Verbesserung der Suche sollten des weiteren zuerst immer die Kindknoten in Richtung des Zielpunktes untersucht werden.

[49] *LIFO*: last in first out

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Heuristische Suchalgorithmen, auf Graphen basierend

---

best first search

Hierbei handelt es sich um einen Suchalgorithmus, der eine *Heuristik* einbezieht, d.h., dass die Suche aufgrund einer Abschätzung zum Zielpunkt in diese Richtung beeinflusst wird.

Prinzipiell entspricht der best first search – Algorithmus dem Dijkstra-Algorithmus, abgesehen von der Tatsache, dass die Sortierung der Open Liste nicht von der Länge vom Startpunkt zum momentanen Suchpunkt bestimmt wird, sondern vielmehr von der Abschätzung des Weges vom Such- zum Zielpunkt. Das Problem ist, dass der Weg um ein Hindernis nicht direkt, sondern wellenförmig berechnet wird. Dementsprechend ist der gefundene Weg nicht unbedingt der Kürzeste.

A* Suche

Der beste Algorithmus, um den kürzesten Weg zwischen zwei Punkten A und B zu berechnen, ist der A*. Der große Vorteil dieser Methode ist, dass der A* sowohl eine abgeschätzte Länge vom momentan besuchten Knoten zum Zielknoten, als auch die Länge vom Startknoten zu diesem, in seine Berechnungen mit einbezieht. Die Abschätzung zum Zielknoten nennt sich Heuristik. Eine Heuristik ist eine „*Arbeitshypothese als Hilfsmittel der Forschung*“[50], d.h. man stellt eine Hypothese die Reststrecke betreffend auf, um die Suche in eine Richtung, nämlich in die des Zielknotens, zu lenken. Mit dieser Methode beschleunigt man die Suche bedeutend, weil nur ein Bruchteil der möglichen Knoten untersucht wird. Der A* basiert, wie der Dijkstra- Algorithmus, auf der Graphentheorie und wird mit folgender Formel beschrieben:

[50] Zitat: Duden Das Fremdwörterbuch, S.309

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Formel:

$f(n) = g(n) + h(n)$

f(n) - die (geschätzte) Gesamtlänge des Weges

g(n) – die tatsächliche Länge vom Startpunkt zum Knoten n

h(n) - heuristische Abschätzung des restlichen Weges von n zum Zielpunkt

Der A*- Algorithmus ist eine Kombination des Dijkstra- und des best first search- Algorithmus. Wie der A* im Detail funktioniert, wird im nächsten Kapitel dargestellt.

Solange die heuristische Abschätzung h(n) zulässig ist, d.h. sie ist niemals größer als die tatsächliche Distanz zum Zielpunkt, wird garantiert der kürzeste Weg zwischen zwei Punkten A und B generiert. Beim A* handelt es sich um den Algorithmus, der am effizientesten mit heuristischen Funktionen arbeitet. Es gibt keinen anderen Algorithmus, der mit der gleichen Heuristik einen besseren Pfad mit weniger Knotenpunkten berechnet.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

5.0

Pathfinding in 2D Welten

Die Graphentheorie

Erläuterung

Die Graphentheorie ist ein Teilgebiet der Mathematik, welches in andere Wissenschaftsbereiche, wie z.B. in die Informatik, Wirtschaftswissenschaft, etc. adaptiert wurde. Graphen ermöglichen die Abbildung von Strukturen in mathematischen Modellen. Prinzipiell werden mit Hilfe der Graphentheorie kürzeste und längste Wege berechnet. Die Länge eines Weges entspricht hierbei der Summe der gewählten Kantenlängen.

Folgende Fragestellungen können, den kürzesten Weg betreffend, mit der Graphentheorie bestimmt werden:

1. Bestimmung des kürzesten Weges zwischen zwei gegebenen Knoten.
2. Bestimmung der p kürzesten Wege zwischen zwei gegebenen Knoten.
3. Bestimmung der kürzesten Wege von einem (Start-)Knoten über alle anderen Knoten des Graphen zu einem gegebenen (Ziel-)Knoten.
4. Bestimmung der kürzesten Wege von jedem Knoten zu jedem anderen Knoten des Graphen.

Die Suche nach dem kürzesten Weg mit Hilfe der Graphentheorie funktioniert folgendermaßen:

Man bestimmt den Startknoten e_0 mit der Entfernung $d = d(e_0) = 0$. Alle Knoten, die direkt über eine Kante mit e_0 verbunden sind, werden nun bezüglich ihrer Länge

überprüft. Man wähle die kürzeste Länge und addiere diese zu d (= $d(e_0)$ + $d(e_0, e_1)$). Dasselbe vollführt man für den gerade ermittelten Knoten e_1 und erhält somit ein neues d = d + $d(e_1, e_2)$. Diese Entfernung d vergleicht man danach mit den anderen möglichen Wegen, ausgehend von e_0, allerdings mit der Einschränkung, dass von dem nächstmöglichen erreichten Knotenpunkt ? e_1 wiederum nur die kürzeste Kantenlänge in die Berechnung einfließt. Ist diese Entfernung kürzer als d, dann ist diese die neue Wegstrecke d. Diese Methode wird wiederholt, bis der Zielknoten erreicht wird.

A*- Algorithmus

Grundlagen

Beim A*- Algorithmus handelt es sich um einen seit 1968 von der akademischen AI - Community verwendeten Algorithmus zur Lösung einer Vielzahl unterschiedlicher Suchprobleme. Das bekannteste Beispiel ist das 15- Felder- Puzzle, in einem 4 * 4 Feld, bei dem man durch Hin- und Herschieben der Felder das Puzzle löst.

Erst im Oktober 1996 eröffnete Bryan Stout erstmals der Game Developer Community die Möglichkeit eines Einsatzes des A* zur Pfadfindung.

Die Heuristik

Der A*- Algorithmus ist, wie bereits erwähnt wurde, eine Suche mittels der *Graphentheorie* unter Berücksichtigung einer *Heuristik*. Die Heuristik ist eine Möglichkeit, die Graphensuche in Richtung eines gewissen Punktes zu leiten, sodass im Gegensatz zu anderen Suchtechniken nur ein Bruchteil der möglichen Punkte kontrolliert werden muss. Prinzipiell schätzt man mit der Heuristik ab, wie groß die Entfernung zwischen dem Zielpunkt und dem aktuellem Punkt ist. Addiert man bei der A* Suche zu der Heuristik die Entfernung zwischen Start- und aktuellem Suchpunkt,

dann kann man abschätzen, indem man diese Gesamtentfernung zu den anderen, bereits Ermittelten abwägt, ob mit diesem Knoten der kürzeste Pfad generiert werden kann. Aufgrund der Tatsache, dass nur sehr wenige mögliche Knoten kontrolliert werden müssen, da gewissermaßen eine Vorstellung für die Distanz zum Zielpunkt vorhanden ist, ist die Suche sehr schnell. Bewegt man sich beispielsweise vom Zielpunkt weg, wobei die heuristische Abschätzung zum Zielpunkt sehr groß wird, dann bedeutet dies, dass die Knoten, die noch weiter vom Zielpunkt entfernt sind, nicht mehr kontrolliert werden müssen. Aufgrund der Heuristik entscheidet man sich für den wahrscheinlich besten Knoten in einer anderen Richtung. Die Methode funktioniert so gut, weil eine heuristische Abschätzung gewählt wird, die kleiner oder gleich der eigentlichen Entfernung ist. Meist handelt es sich um die Luftlinie vom aktuellen Punkt zum Zielpunkt. Dies hat zur Folge, dass sich die Suche sehr stark in Richtung Zielpunkt bewegt. Im Gegensatz dazu kontrolliert man beim Dijkstra, weil die Richtung nicht beachtet wird, kreisförmig alle möglichen Suchpunkte. Diese Vorgehensweise ist, aufgrund der weitaus größeren Suchmenge, vergleichsweise sehr langsam.

Schon hier sollte darauf hingewiesen werden, dass die Heuristik wie ein Magnet funktioniert und dass es durchaus schwierig ist, den A* von seiner eingeschlagenen Richtung abzubringen, was für die in der Diplomarbeit verwendete Methode notwendig ist.

*Erläuterung des A**

Da nun das Prinzip der Graphensuche in Verbindung mit der Heuristik bekannt ist, kann man sich der Arbeitsweise des A* widmen.

Prinzipiell überwacht der A* zwei Kontrolllisten. Die eine Liste beinhaltet bereits kontrollierte (Weg-)Knoten, diese Liste nennt sich *Closed List*. Die andere beinhaltet dementsprechend noch nicht kontrollierte Knoten und nennt sich *Open List*. Zu Beginn der Suche ist die Closed List leer. Die Open List beinhaltet nur den Startpunkt.

Bei jedem Schritt benutzt der Algorithmus für seine Berechnungen den vielversprechendsten Knoten der Open List und entfernt diesen aus der Liste. Der vielversprechendste Knoten ist derjenige, der auf dem Weg mit der kürzesten (geschätzten) Gesamtentfernung zum Zielpunkt liegt. Zu Beginn kann dies nur der Startknoten sein. Handelt es sich bei diesem Knoten nicht um den Zielknoten, dann werden alle Nachbarknoten, die mit diesem Knoten direkt verbunden sind, überprüft. Handelt es sich bei den Nachbarknoten um solche, die noch nicht kontrolliert wurden, dann werden sie in Open platziert. Befinden sie sich bereits in Open oder Closed, dann werden alle notwendigen Informationen der Knoten, wenn sie einen kürzeren Pfad garantieren, aktualisiert oder, wenn sie sich in der Closed List befinden, der Open List hinzugefügt und aus der Closed List entfernt. Sollte die Open List leer sein bevor der Zielpunkt gefunden wurde, dann gibt es keinen Pfad vom Start- zum Zielknoten.

Der vielversprechendste Knoten zu einem bestimmten Zeitpunkt der Suche, der aus der Open List gewählt wird, ist, wie erwähnt, derjenige, der auf dem kürzesten Weg durch die Lokation liegt. Jeder Knoten n, bei den Knoten handelt es sich im eigentlichen Sinn um Punkte, beinhaltet Informationen, die Aufschluss über den Weg geben. Eine Information ist z.B. die Länge des Pfades, der vom Startpunkt zu n führt; im Pseudocode wird dies *CostFromStart(n)* genannt. Weitere Informationen sind die, anhand der Heuristik, abgeschätzte restliche Länge des Weges von n zum Zielpunkt, die man *CostToGoal(n)* nennt und die gesamte Pfadlänge *TotalCost(n) (= CostFromStart(n) + CostToGoal(n))*. Anhand dieser geschätzten gesamten Pfadlänge

werden die nächsten Knoten ausgewertet. Weiterhin beinhaltet jeder Knoten einen Zeiger auf seinen Elternknoten, um, wenn ein Pfad vom Start- zum Zielpunkt gefunden wird, den Weg generieren zu können.

Der gesamte Pfad jeden Knotens wird anhand folgender Formel dargestellt:

Formel:

$f(n) = g(n) + h(n)$

f(n) - die (geschätzte) Gesamtlänge des Weges

g(n) – die tatsächliche Länge vom Startpunkt zum Knoten n

h(n) - heuristische Abschätzung des restlichen Weges von n zum Zielpunkt

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Pseudocode:

```
Open: priority queue von nicht untersuchten Suchknoten
Closed: Liste von untersuchten Suchknoten

AStarSearch( Point ptStart, Point ptGoal)
{
        Open und Closed leeren

        // Startknoten initialisieren
        StartNode.ptLoc = pt.Start
        StartNode.CostFromStart = 0
        StartNode.CostToGoal = PathCostEstimate(ptStart, ptGoal)
        // PathCostEstimate(..) ist die Heuristikberechnung
        StartNode.parent = null
        Push StartNode nach Open

        // Suche durchführen, bis ein Weg gefunden wurde oder Open leer ist
        while(Open ist nicht leer)
        {
                pop vielversprechendsten Knoten Node von Open
                if(Node ist Zielpunkt ptGoal)
                        Generiere Pfad und beende Suche
                else
                {
                        for(jeden neuen Nachbarknoten NewNode von Node)
                        {
                                NewCost = Node.CostFromStart
                                NewCost += TraverseCost(Node, NewNode)
                                // TraverseCost(..) berechnet Entfernung zwischen
                                // den Nachbarknoten
                                bTest =  NewNode.TotalCost <= (NewCost +
```

Autor: René Greulich Matrikelnr.: 110 18781
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke Zweitprüfer: Prof. Dr. E. Ehses

```
                    PathCostEstimate(NewNode.ptLoc, ptGoal))
                if((NewNode ist in Open oder Closed) and (bTest))
                    continue
                else
                {
                    NewNode.parent = node
                    NewNode.CostFromStart =NewCost
                    NewNode.CostToGoal =
                        PathCostEstimate(NewNode.ptLoc, ptGoal)
                    NewNode.TotalCost = NewNode.CostFromStart
                            + NewNode.CostToGoal
                    if(NewNode ist in Closed)
                            Lösche es aus Closed
                    if(NewNode ist in Open)
                            führe update durch
                    else
                            NewNode zu Open hinzufügen
                }
            }
        }
        Node zu Closed hinzufügen
    }
    return Fehler
}
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Eigenschaften des A*- Algorithmus

1. Wenn es einen Pfad von einem Punkt A nach B gibt, dann wird dieser auch gefunden.
2. Solange CostToGoal(n) zulässig ist, d.h. CostToGoal(n) muss immer kleiner gleich der tatsächlichen kürzesten Länge des Pfades sein, wird immer ein optimaler Pfad generiert.
3. Der A* arbeitet am effektivsten mit einer gegebenen Heuristik. Es gibt keinen anderen Suchalgorithmus der, anhand der gleichen Heuristik, einen derart perfekten Pfad mit weniger Knotenpunkten errechnet.

Hierzu gibt es eine entsprechende Beweisführung von Nils J. Nilsson[51].

Detailerläuterung

Nachdem der A* erläutert wurde, soll noch darauf eingegangen werden, wie die für die Umsetzung des A* wichtige Abbildung der Lokation aussehen kann, wie die Nachbarknoten ermittelt werden sollten, was der allgemeine, relative und absolute Aufwand ist, und wo die Schwäche des A* liegt.

Verwendung des A* zur Pfadplanung in Computerspielen

Bis zu diesem Zeitpunkt wurde nur die Position einer Entität für die Pfadsuche betrachtet. Unter gewissen Umständen, abhängig von der Art des Spiels, ist gegebenenfalls auch die *Orientation*[52] oder die *Velocity*[53] einer Entität von Bedeutung. Handelt es sich bei der Entität z.B. um ein Fahrzeug, ist die Bewegungsfreiheit der Entität in der Regel eingeschränkt. Ein Fahrzeug kann sich, wie es in Spielen oftmals vorkommt, nur vorwärts bewegen und große Kurven fahren, die wiederum abhängig von der Geschwindigkeit des Fahrzeugs sind. Um rückwärts zu fahren, müssen die

[51] *Nils J. Nilsson:* Nilsson ist einer der Erfinder des A*. Die Beweisführung kann in seinem Buch „Artificial Intelligence: A New Synthesis“ nachgelesen werden.

[52] *Orientation*: Richtung aus der eine Entität betrachtet wird

[53] *Velocity*: Geschwindigkeit einer Entität

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Fahrzeuge meist erst einmal stoppen. Abhängig von der Orientation und Velocity kann es von Vorteil sein, die möglichen Pfade so vorzuplanen, dass bestimmte Routen durch ein Terrain oder um Hindernisse herum, nicht passierbar sind. Man kann sich vorstellen, wie ein Jump and Run- Spiel für Kinder angenommen wird, dass graphisch perfekt umgesetzt ist, aber hinsichtlich der Steuerung, aufgrund zu komplizierter Wegführungen, die Fähigkeiten der Kinder überfordert.

Um dieser Problematik zu entgehen, und um eine gute Möglichkeit der Abbildung einer Lokation bzw. Welt zu haben, ist es notwendig, diese in verschiedene Laufbereiche einzuteilen.

Abbildung 14:
Möglichkeiten der Aufteilung einer Lokation

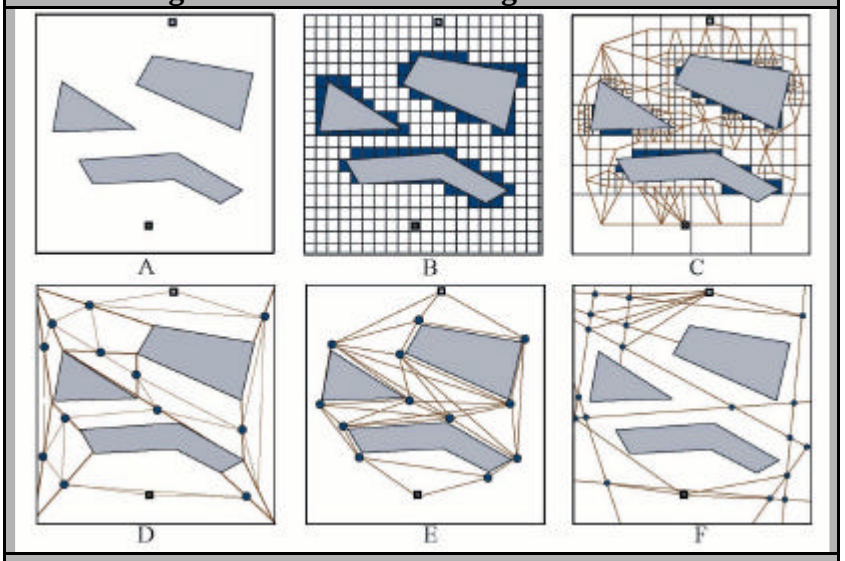


Bild A zeigt eine Lokation, die drei Hindernisse beinhaltet. Die Bilder B – F zeigen fünf verschiedene Möglichkeiten, die freie Fläche zwischen den Hindernissen zu verwalten.

Bild B: *Rectangular Grid* – Gitternetz von Quadraten: Die einfachste Möglichkeit eine Lokation zu partitionieren ist eine Aufteilung in gleichgroße Quadrate. Die zu untersuchenden Punkte eines Quadrates sind entweder das Zentrum oder die Ecken.

Bild C: *Quadtree* – Quadratbaum: Eine weitere Möglichkeit die Laufpfade zu verwalten ist die Aufteilung der Lokation in Quadrate verschiedener Größen. Der Quadratbaum unterteilt rekursiv jedes Quadrat in vier kleinere Vierecke, bis jedes Quadrat ungefähr eine gleich große Fläche hat. Auch hier können die Kontrollpunkte entweder das Zentrum oder die Ecken des Rechteckes sein.

Bild D: *Convex Polygons* – konvexe Polygone: Eine andere, etwas komplexere, aber robustere Möglichkeit der Aufteilung der Lokation sind Polygone. In der Regel gibt es eine solche Aufteilung bereits in 3D *Engines*[54], weil dort alle Objekte aus Polygonen bestehen. Somit kann die vorhandene Welt, so wie sie ist, zur Pfadsuche verwendet werden. Wenn es kein *Mesh*[55] gibt oder das vorgegebene Mesh für die Pfadsuche unvorteilhaft ist, kann mit verschiedenen Methoden, wie z.B. mit *C-cells,* die Partitionierung selbst durchgeführt werden. Bei den C-cells werden die einzelnen vorhandenen sichtbaren *Vertexes*[56] eines Objektes mit der nächstbesten Vertex verbunden. Hierbei wird der Raum durch die Verbindungslinien unterteilt. Eine andere Methode wäre die *maximum-area decomposition* oder *Navigation Meshes*. Bei den Polygonen sind die Kontrollpunkte das Zentrum des Polygons und/ oder verschiedene Punkte des Umrisses.

Bild E: *Points of Visibility* – Sichtbarkeitspunkte: Hierbei werden um die Objekte bzw. Hindernisse Punkte gelegt, die für diese eine Umrandung darstellen. Diese Form der Partitionierung verwendet man hauptsächlich bei der *obstacle avoidance*[57]. Die Punkte dienen dazu, dass kurz vor einer möglichen Kollision mit dem Objekt, auf diese reagiert werden kann.

---

[54] *engine*: die Ansammlung aller c/c++ Bibliotheken, die die Funktionalität eines Spiels ausmacht.

[55] *Mesh*: Gittermodell eines Objektes, alle Punkte, die ein Modell aufspannen.

[56] *Vertex:* Dreieck; ist die kleinste Einheit für eine Fläche im 3D Bereich; Standardobjekt.

[57] *obstacle avoidance*: Ignorierung des Hindernisses.

Bild F: *Generalized Cylinders*: Der Raum zwischen den Hindernissen kann als 2D Zylinder aufgefasst werden. Zwischen zwei benachbarten Hindernissen und den Grenzen der Lokation wird nun eine zentrale Achse festgelegt. Die Schnittpunkte der Achsen sind die Punkte, die während der Suche kontrolliert werden.

In der Engine, die für das Beispielprogramm verwendet wird, werden die Bereiche zwischen den Hindernissen, d.h. alle Bereiche auf denen gelaufen werden kann, in *Areas*[58] aufgeteilt. Nur innerhalb dieser Areas, es kann sich auch um *Pfade*[59] handeln, findet eine Suche statt. Hierbei entsteht ein sehr geringer Speicheraufwand, weil sich die Areas durch wenige Eckpunkte definieren. Allerdings müssen für jeden Suchpunkt im schlechtesten Fall alle Areas kontrolliert werden.

Das Problem ist, dass es keine Generalisierung für die Aufteilung von Lokations gibt. An jedes Spiel bzw. an jede Spieleengine muss die Aufteilung der Lauflokations angepasst werden. Hierbei muss darauf geachtet werden, dass genug Partitionen vorhanden sind, um eine genaue Wegberechnung zu gestalten, allerdings dürfen auch nicht zu viele generiert werden, um die Suche nicht zu langsam zu gestalten.

[58] *area*: Eine Area ist ein Vieleck mit mindestens vier Eckpunkten, zwischen denen gelaufen werden kann.

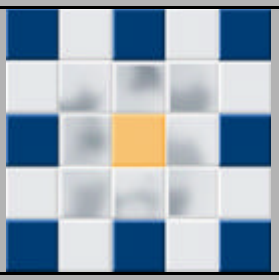
[59] *Pfad*: ein Pfad ist eine Linie, auf der gelaufen werden kann.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Ermittlung der Nachbarknoten/ -punkte eines Punktes

Um die A* Suche umzusetzen, muss unabhängig von der verwendeten *map representation*[60] eine effiziente Methode entwickelt werden, wie die Nachbarknoten eines Suchknotens ermittelt werden können. In der Regel hat ein Punkt im 2D – Bereich 4 oder 8 Nachbarn, abhängig davon, ob diagonale Nachbarn in die Pfadfindung miteinbezogen werden.

Abbildung 15: Nachbar- /Nachfolge-/ Kindknoten



Gelb: aktueller Punkt

Graumeliert: 8 Nachbarknoten

Normalerweise berechnen sich die (graumelierten) Nachbarn eines Punktes n(x,y) [gelb] nach folgendem Modell:

- Oberhalb: $n_1(x, y - 1)$
- oben rechts: $n_2(x + 1, y - 1)$
- rechts: $n_3(x + 1, y)$
- unten rechts: $n_4(x + 1, y + 1)$
- unten: $n_5(x, y + 1)$
- unten links: $n_6(x - 1, y + 1)$
- links: $n_7(x - 1, y)$
- oben links: $n_8(x - 1, y - 1)$

[60] *map representation*: Abbildung der Lokation(aufteilung)

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Aufwand[61]

Man unterscheidet den *absoluten*, den *relativen* und den *allgemeinen Aufwand*. Der absolute Aufwand ist die Länge vom Startpunkt zum aktuellen Suchpunkt (CostFromStart). Der relative Aufwand ist die abgeschätzte Restlänge vom aktuellen Suchpunkt zum Zielpunkt (CostToGoal, Heuristik oder Estimate). Der allgemeine Aufwand ist die Summe dieser beiden Aufwendungen.

allgemeiner Aufwand

Der allgemeine Aufwand beschreibt immer, was minimiert werden soll. In der Regel ist dies die kürzeste Distanz zwischen zwei Punkten. Es könnte auch der Tankverbrauch zwischen zwei Städten sein, der während einer Reise entsteht.

Verschiedene Grundvoraussetzungen des Terrains können den Aufwand beeinflussen. Von A nach B zu reisen, könnte wesentlich vorteilhafter sein, als den umgekehrten Weg zu gehen. Vielleicht hat man von A nach B Rücken- und dementsprechend in der anderen Richtung Gegenwind. Es muss ein System geben, unabhängig von der Relativierung der Aufwendungen, mit dem schnell herausgefunden werden kann, wovon die Aufwendungen abhängen und wie groß diese sind.

relativer Aufwand – Estimate - Heuristik

Um zu garantieren, dass der perfekte Pfad gefunden wird, ist es eine Grundvoraussetzung, dass die Distanz nicht überschätzt wird. Aus diesem Grund wird in der Regel die *Luftlinie* zwischen zwei Punkten zur Ermittlung der Entfernung bzw. Heuristik verwendet, die mit dem *Satz des Pythagoras* berechnet werden kann.

Eine Grundvoraussetzung für ein gutes Pathfindingsystem ist nicht nur, dass der kürzeste oder beste Weg gefunden, sondern dass dieser auch in kürzester Zeit berechnet wird. Setzt man die Heuristik auf 0, dann breitet sich die Suche nach dem besten Pfad, da diese durch keine heuristische Information geleitet wird, in alle

[61] *Aufwand:* Fachbegriff ist cost

Richtungen der Lokation aus. Dies hat zur Folge, dass der Suchumfang sehr groß ist. Daher muss darauf geachtet werden, dass die Heuristik nicht zu *gering gewählt wird*[62].

In manchen Fällen wird, entgegen der Eigenschaften des A*, um eine sehr schnelle Suche zu generieren, die Heuristik leicht *überschätzt*[63]. Das bedeutet, dass im Fall einer Überschätzung keine Heuristik gewählt wird, die kleiner als die tatsächliche Distanz ist, sondern eine die größer ist, sodass eine noch stärkere Beeinflussung der Suche in Richtung Ziel stattfindet. Dies hat zur Folge, dass der Pfad nur beinahe der Kürzeste ist. Es sollte darauf hingewiesen werden, dass eine Überschätzung der Heuristik, wenn sie unkontrolliert ist, die Performanz sehr stark belastet.

Schwäche des A*

Obwohl es sich beim A*- Algorithmus um den besten Suchalgorithmus handelt, müssen bestimmte Dinge beachtet werden, um eine Verschwendung der Ressourcen zu vermeiden. Bei sehr großen Maps, bei denen in Open und Closed Hunderte oder Tausende von Knoten verwendet werden, kann es auf Systemen mit wenig Speicher, wie z.B. bei Konsolen, passieren, dass mehr Speicher gebraucht wird, als vorhanden ist. Bei allen Systemen kann es passieren, dass zu viel CPU Zeit verwendet wird.

Am ineffizientesten ist die Verwendung des A* bei der Berechnung von Pfaden zwischen zwei Punkten, bei denen es keine Verbindung gibt. Um dieses Problem zu umgehen ist es ratsam, eine Voranalyse durchzuführen.

[62] *gering gewählt wird* : Fachbegriff ist underestimated

[63] *überschätzen:* Fachbegriff ist overestimated

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

6.0

Optimierung des A*

Optimierung

Nachdem die Grundlagen für die A* Pfadplanung vermittelt worden sind, besteht die Aufgabe dieses Kapitels darin, zu zeigen, wie das Pathfindingsystem perfektioniert werden kann, um einerseits ein realistisches Verhalten zu simulieren und andererseits die Performanz positiv zu beeinflussen. Beide Aspekte dienen dazu, dem Spieler eine bessere Grundvoraussetzung zum Spielen zu liefern. Zum einen bewirkt die Performanz- und Geschwindigkeitsoptimierung eine bessere Bildwiederholungsrate, dies hat zur Folge, dass die Grafikdarstellung flüssiger aussieht und alle internen Tasks, wie die AI, schneller verarbeitet werden. Zum anderen wird der Spieler durch ein schönes Verhalten der Charaktere verwöhnt.

Ästhetische Optimierung

Ziel der ästhetischen Optimierung

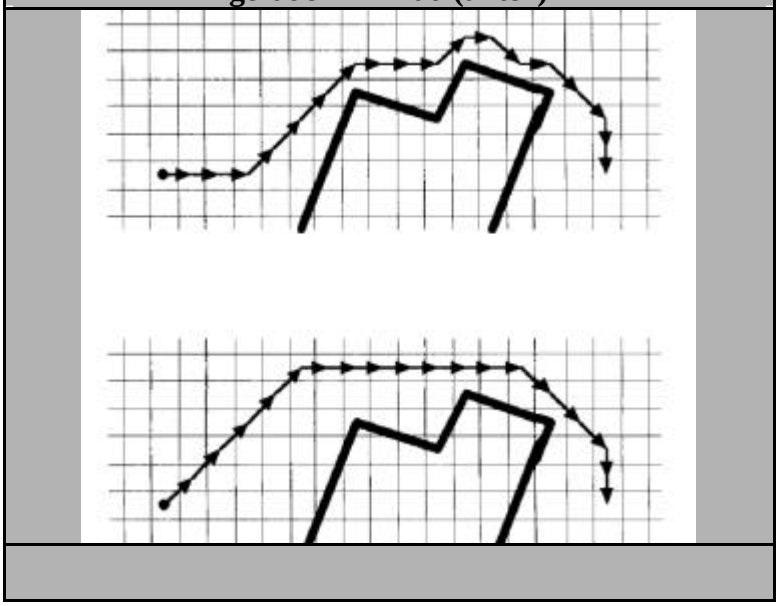
Die ästhetische Optimierung des A* bewirkt, dass natürlichere Pfade erstellt werden. Primäres Ziel der ästhetischen Optimierung ist, die Pfade zu begradigen und den Kurvenverlauf geschmeidiger darzustellen. Abhängig von der Wahrnehmung werden bei der ästhetischen Optimierung unterschiedliche Methoden verwendet. In Situationen, die der Spieler direkt wahrnimmt, geschieht dies anhand performanzintensiver Methoden. Verwendete Methoden sind die Techniken des

straight und des *smooth paths*. Manchmal bedarf es aber nur einer abgeschwächten ästhetischen Optimierung. In diesem Fall verwendet man die *hierarchische Suche*.

Straight Paths – Ein gerader Weg

In der Regel entwickelt der A*- Algorithmus einen Pfad, der sehr zickzackmäßig verläuft. Grund dafür ist, dass, sobald ein kürzester Pfad oder Abschnitt erkannt worden ist, dieser direkt in die weitere Suche des A* eingeht.

Abbildung 16:
Vergleich normaler A* Pfad (oben) und gerader A* Pfad (unten)



Das Problem, das dadurch entsteht, ist, dass dieser Pfad nicht natürlich aussieht und somit die Glaubwürdigkeit der AI in Frage gestellt werden könnte. Betrachtet man nun die oberen beiden Pfade etwas genauer, dann stellt man fest, dass sie exakt die gleiche Länge haben. Das Problem ist, dass der normale A* nicht feststellen kann, ob es zwei oder mehrere Nachbarknoten gibt, die die gleiche Länge ermitteln. Der A* arbeitet mit dem erstbesten, vorteilhaftesten Knoten, ohne diesen in Bezug zu anderen abzuwägen.

Um dem A* zu vermitteln, dass es einen besseren Knoten mit der gleichen Länge gibt, muss man eine weitere Wertung mit in die Gesamtlänge des Pfades einfließen lassen. Normalerweise würde man der Gesamtlänge eines berechneten

Pfades einen *Strafwert*[64] von der Hälfte der Distanz zum Nachbarknoten hinzufügen, wenn der ermittelte Nachbarknoten nicht der Richtung des aktuellen Suchknotens folgt. Bei einer regular grid[65] bewirkt allerdings schon ein *Penalty* von 10^{-7}, dass der geradeste Pfad ermittelt wird.

Smooth Paths – Pfade mit geschmeidiger Kurvenführung

Auch wenn die mit dem A* ermittelten Pfade begradigt worden sind, haben sie immer noch die Eigenart sehr spitze Kurven aufzuweisen, was dazu führt, dass die Bewegungen der Charaktere eher denen von Robotern, als denen von Menschen ähneln.

Es gibt eine Methode, die normalerweise im Bereich der Echtzeitberechnung von Computergraphiken verwendet wird, mit deren Hilfe eine Kurve berechnet werden kann, die zwischen zwei Punkten verläuft. Hierbei handelt es sich um die *Catmull Rom Formel*. Das Faszinierende der *Catmull Rom Formel* ist, im Gegensatz zu der Berechnung von *Bézier Kurven*[66], dass die ermittelte Kurve tatsächlich durch die zur Errechnung der Kurve notwendigen Kontrollpunkte verläuft. Bei den Bézier *Kurven* kann eine neue Kurve unabhängig von den verwendeten Punkten entstehen.

Die *Catmull Rom Formel* benötigt vier Punkte, mit denen sie eine geschmeidige Kurve zwischen dem zweiten und dritten Punkt berechnet. Um die Kurve zwischen dem ersten und zweiten Punkt des ermittelten Pfades zu berechnen, benutzt man zweimal den ersten Punkt, den zweiten und dritten Punkt. Um die Kurve zwischen den letzten beiden Punkten eines Pfades zu berechnen, benötigt man den vorvorletzten, den vorletzten und zweimal den letzten Punkt.

---

[64] Strafwert: Fachbegriff ist Penalty

[65] *regular grid:* rectangular grid

[66] *Bézier Kurven*: bei kubischen Bézier Kurven werden, anhand von vier Kontrollpunkten Kurven erstellt, die durch den ersten und vierten Punkt gehen und sich den Punkten zwei und drei annähern.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Die *Catmull Rom Formel* berechnet, wie erwähnt, einen Punkt zwischen dem zweiten und dritten Punkt. Sie benötigt einen Prozentsatz u im Bereich 0 und 1, der besagt in welchem prozentualen Abstand der Punkt zum zweiten Punkt liegen soll. Wenn u = 0 ist, dann errechnet die Formel den zweiten, wenn u = 1, den dritten Punkt. Das Faszinierende ist, dass die Formel sowohl 2D, als auch 3D Punkte verarbeiten kann.

Catmull Rom Formel:

```
SplineOutputPoint = Point1 * (-0.5f*u*u*u  +      u*u   - 0.5f*u) +
                    Point2 * ( 1.5f*u*u*u   -  2,5u*u  + 1.0f)     +
                    Point3 * (-1.5f*u*u*u   +  2,0u*u  + 0.5f*u)+
                    Point4 * ( 0.5f*u*u*u   -  0,5u*u)
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Aufgrund der Tatsache, dass wieder einmal die Rechenzeit sehr entscheidend ist, ist es ratsam, die Formel für ein bestimmtes Intervall, z.B. u = 0.0, u = 0.25, u = 0.5, u = 0.75, u = 1.0, im Voraus zu berechnen:

Vorberechnete Catmull Formel:

1. Bereich:

   u = 0

   $SplineOutputPoint_{0.0}$ = Point2

2. Bereich:

   u = 0.25

   $SplineOutputPoint_{0.25}$ = Point1 * -0.0703125f + Point2 * 0.8671875f +
   Point3 * 0.226525f + Point4 * -0.0234375f

3. Bereich

   u = 0.5

   $SplineOutputPoint_{0.5}$ = Point1 * -0.00625f + Point2 * 0.5625f +
   Point3 * 0.5625f + Point4 * -0.0625f

4. Bereich

   u = 0.75

   $SplineOutputPoint_{0.75}$ = Point1 * -0.0234375f + Point2 * 0.226563f +
   Point3 * 0.8671875f + Point4 * -0.0703125f

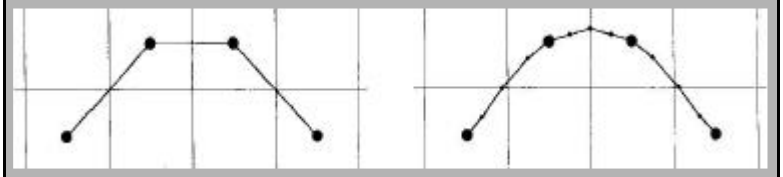
5. Bereich

   u = 1

   $SplineOutputPoint_{1.0}$ = Point3

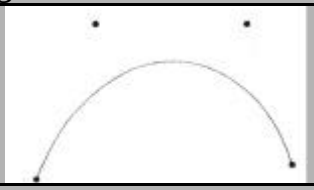
Bedenkt man, dass man mit dieser Methode einen viermal so großen Pfad errechnet, ist schon ersichtlich, wie geschmeidig und rund der Pfad im Gegensatz zum ursprünglichen ist.

Abbildung 17:
normaler und geschmeidiger Pfad



Geschmeidiger Pfad anhand der Catmull Formel

Abbildung 18:
Geschmeidiger Pfad anhand von Bézier Kurven



Hierarchische Pfadsuche

Lokales und universelles Pathfinding

Bei der Pfadfindung muss man verschiedene Pfadfindungssituationen unterscheiden. Die lokale Pfadfindung ermittelt einen Pfad, dessen Punkte sich in einer Lokation befinden. Ziel dieser Pfadfindung ist es, einen optisch wunderschönen Pfad zu generieren, der das Auge des Betrachters verwöhnt. Diese Pfadfindung findet für alle NPCs und den PC statt, wenn sie sich in der momentan sichtbaren Lokation befinden (2D) oder im sichtbaren Bereich eines Levels[67] (3D). Die lokale Pfadfindung verbraucht sehr viel Speicher und Rechenleistung, weil damit sehr genaue, gerade und formschöne Pfade berechnet werden. Eine zweite Pfadfindungssituation betrifft das universelle Pathfinding, dass nur NPCs betrifft. Ein Spiel hat in der Regel weit mehr als nur eine Lokation. Ein Spiel umfasst z.B. ein kleines Dorf. Dieses Dorf besteht

[67] *Levels*: Adäquat zur Lokation im 3D Bereich

dann vielleicht aus einer Lokation Marktplatz, eine Lokation Platz vor dem Haus des Bürgermeisters, einer Lokation Bibliothek, einer Lokation Bäckerei, etc. Diese Lokations beschreiben nun das Universum, die Welt dieses Spiels, und sind über bestimmte Verbindungspunkte miteinander verbunden. Der Marktplatz ist vielleicht mit dem Platz vor dem Haus des Bürgermeisters verbunden. Von dort aus kommt man in die Bibliothek und zur Bäckerei. Die Verbindungspunkte geben an, wohin ein automatisierter NPC laufen muss, um von dort aus in die nächste Lokation zu gelangen. Das universelle Pathfinding dient dazu den Weg eines NPCs über ein oder mehrere Lokations durch die Welt, zu generieren. Ein Weg führt vielleicht von der Bibliothek über den Vorplatz beim Bürgermeister zum Marktplatz. Es sei darauf hingewiesen, dass man sich als Spieler nur in einer dieser Lokations befindet, d.h. nur eine Lokation ist zu einem Zeitpunkt sichtbar. Aufgrund der Tatsache, dass die automatisierten NPCs ihrem geregelten Tagesablauf nachgehen müssen, bewegen sie sich zwischen den Lokations. Diese Pfade werden anhand des universellen Pathfindings berechnet. Sie werden nicht so genau berechnet, wenn sich der entsprechende NPC nicht in derselben Lokation wie der PC befindet, weil der NPC dann nicht sichtbar ist. Treffen die NPCs nun auf einen PC, d.h. sie werden für den Spieler sichtbar, dann wird ein genauerer Pfad ermittelt, die Kollisionen abgefragt, etc. Dies hat einen Mehraufwand an Rechenleistung zur Folge.

Hierarchische Pfadsuche

Eine wichtige Technik der Spiele AI in Bezug auf die Pfadfindung ist das *hierarchische Pathfinding*, dass sowohl für das universelle, als auch für das lokale Pathfinding benutzt werden kann. Mit dieser Methode wird eine Route in zwei Schritten berechnet. Zuerst errechnet man einen ungefähren Pfad zwischen den Verbindungspunkten der zu überwindenden Lokations oder der Areas innerhalb einer Lokation. Indem erst einmal nur ein approximaler Pfad berechnet wird, findet eine enorme Rechenzeitersparnis statt. Um danach den idealen Pfad zu berechnen, muss anhand der ermittelten Punkte die Berechnung des eigentlichen Pfades folgen.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

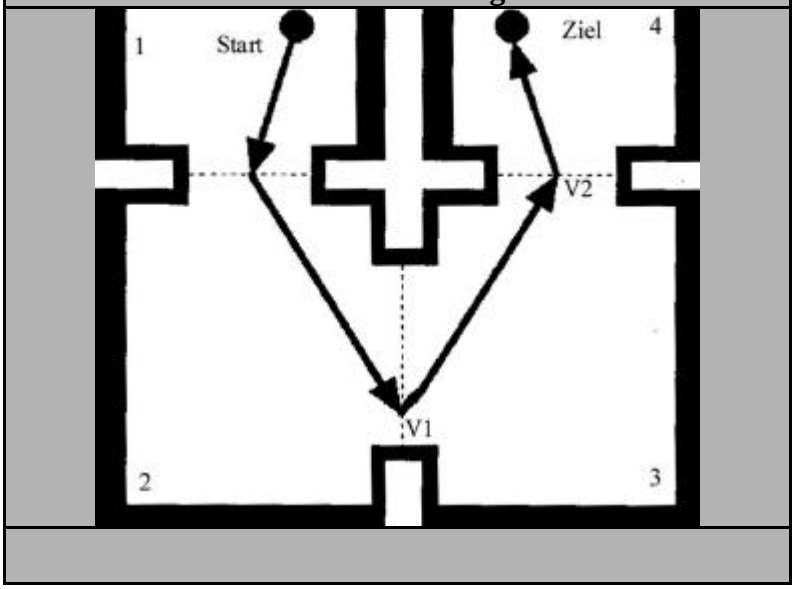
Indem bei der Suche des Pfades der allgemeine, vorgenerierte Weg berechnet wird, wird immer ein idealer Weg zwischen zwei Verbindungspunkten gefunden. Wie die Suche im Detail funktioniert lässt sich am besten an einem Beispiel zeigen.

Beispiel:

Aufgabe: Ermittlung des besten Weges zwischen zwei Punkten über mehrere Areas.

1. Berechne den allgemeinen bzw. generellen Weg vom Start zum Ziel über die Areas 1, 2, 3, 4.

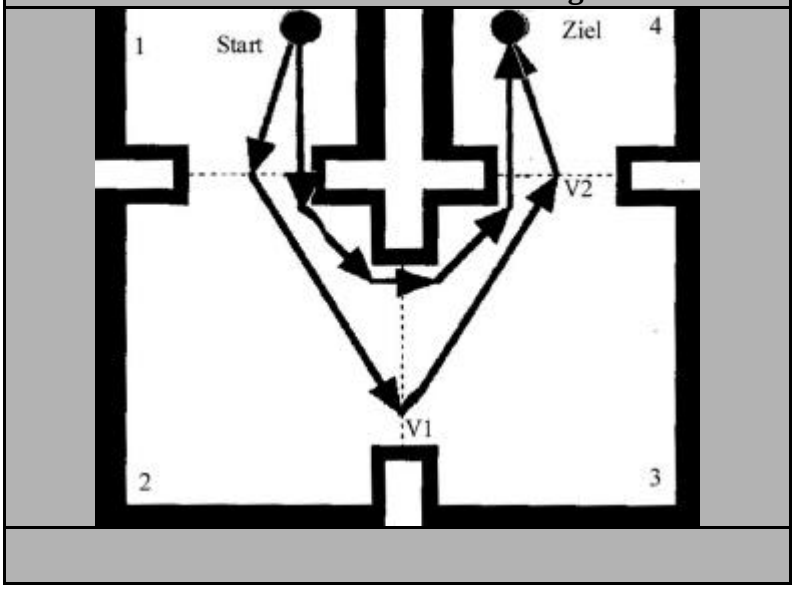
Abbildung 19:
Genereller Weg



2. Berechne nun den kürzesten Weg vom Start zum Verbindungspunkt V1.
3. Laufe, bis der Raum 2 betreten wird.
4. Verwerfe den restlichen Pfad und berechne nun den kürzesten Weg von der aktuellen Position zum Verbindungspunkt V2.
5. Laufe, bis der Raum 3 betreten wird.
6. Verwerfe den restlichen Pfad und berechne nun den kürzesten Weg von der aktuellen Position zum Zielpunkt.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Abbildung 20:
Genereller und idealer Weg



Mit dieser Methode wird nicht der geradeste und kürzeste Weg berechnet, aber die Technik sollte in der Regel auch nur dann angewendet werden, wenn ein NPC einen sehr langen Weg zurücklegen soll, wie z.B. von der Bibliothek zum Marktplatz. Bei einem langen Weg ist die Ungenauigkeit relativ. Der große Vorteil dieser Methode ist, dass die genauere Berechnung des Weges nur dann durchgeführt werden muss, wenn der NPC sichtbar wird. Dies hat eine extreme Entlastung der Rechenzeit zur Folge.

Ein prinzipielles Problem der *hierarchischen Suche* ist, dass der Charakter, wenn er die Lokation oder Area betritt, eine kurze Pause einlegt, um den neuen Pfad zu berechnen. Aus diesem Grund sollte man dafür sorgen, dass der neue Pfad kurz vor dem Betreten der Lokation bzw. Area berechnet wird.

Fazit

Auch in der Spieleprogrammierung versucht man, die Mensch- Computer-Interaktion so perfekt wie möglich zu gestalten. Aus diesem Grund wirken Pausen, die aufgrund des hohen Aufwandes der Pathfindingberechnung entstehen, sehr schlecht. Um dem entgegenzuwirken, muss man dem Spieler das Gefühl vermitteln, dass diese

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Pausen gar nicht existieren, d.h. auf seine Aktion muss direkt reagiert werden. Klickt der Spieler z.B. in die Lokation, damit sich sein PC zu diesem Punkt bewegt, dann muss eine direkte Reaktion des Systems folgen. Ein plumpes Feedback ist, einen kurzen Sound abzuspielen, um dem Spieler mitzuteilen, dass eine Aktion folgen wird. Eine bessere Methode ist eine Reaktion der Steering- Engine. In dem Moment nach dem Klick könnte sich der PC schon mal in die ungefähre Laufrichtung drehen, sodass der Eindruck entsteht, dass das Laufen schon begonnen hat. Somit gewinnt man etwas Zeit, den Weg zu berechnen. Sollten die Berechnungen des Pathfindings allgemein sehr langsam sein, dann könnte man den Charakter auch schon mal in die ungefähre Richtung laufen lassen. Von dieser Methode ist abzuraten, weil die Wahrscheinlichkeit eine falsche Richtung zu wählen sehr groß ist.

Das Problem einer Reaktion wird noch schwieriger, wenn man eine zusammengehörige Gruppe bewegen muss. Die Rechenzeit hierfür ist, wie man sich vorstellen kann, sehr groß. Man hat zwei Möglichkeiten, um diese Situation zu entschärfen. Entweder sorgt man dafür, dass immer nur ein paar wenige Charaktere gleichzeitig den Weg nacheinander berechnen, was dazu führt, dass der Eindruck entsteht, dass die Charaktere nacheinander losgehen, oder man ermittelt den wichtigsten Charakter einer Gruppe, berechnet für diesen den Weg und lässt alle anderen ihm folgen.

Das höchste und schwierigste Ziel eines natürlichen Pathfindingsystems ist, das Pathfinding so realistisch wie möglich zu gestalten, um dem Spieler den Eindruck zu vermitteln, er befände sich in einer echten Welt. Die vorgestellten Techniken sind die besten Methoden, dieses Ziel zu verfolgen. Mit ihrer Hilfe werden geradere und geschmeidigere Wege errechnet, was einen großen Einfluss auf die Wahrnehmung des Spiels hat. Da die Rechenleistungen der heutigen Personalcomputer für die Graphik- Task- und AI- Verarbeitung noch nicht vollkommen ausreichen, muss ein Konsens bezüglich des Rechen- und des Speicheraufwands gefunden werden. Aus diesem Grund werden jetzt performanzverbessernde Techniken vorgestellt.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Speicher-/ Performanz-/ Geschwindigkeitsoptimierung

*Optimierung des A**

Wie bereits festgestellt, besteht beim A* die Gefahr der Ressourcenverschwendung. Dies kann dazu führen, dass mehr Speicher oder Rechenzeit benötigt wird, als überhaupt vorhanden ist. Aus diesem Grund sollte bei der Entwicklung einer AI Engine darauf geachtet werden, dass die Engine nicht nur auf leistungsfähigen PC Systemen funktioniert, sondern auch auf den gängigen Konsolen wie PS2[68] und XBox[69]. Die Playstation 2 hat z.B. 32 MB Arbeits- und 4 MB Videospeicher, sowie eine 128-bit CPU mit einer Taktfrequenz von ca. 295 MHz. Wie auf der Game Developer Conference festgestellt wurde, bekommt eine AI Engine ca. 35 - 38% der zur Verfügung stehenden CPU- Zeit zugeschrieben. Die Tatsache, dass Konsolen, im Gegensatz zum PC, mit sehr wenig Speicher ausgestattet sind, und die Feststellung, dass die AI Programmierung nur einen Teil der Rechenleitung verwendet, zeigt, wie vorbildlich mit dem Speicher und der Rechenleistung umgegangen werden muss. Aufgrund dieser Problematik scheitern viele Engines an der Portierung vom PC zur Konsole.

„Im Prinzip kann die Leistungsfähigkeit eines Algorithmus oft mit beliebig hoher Genauigkeit analysiert werden, wobei sich Einschränkungen nur aus der Unsicherheit hinsichtlich der Leistungsfähigkeit des Computers oder aus Schwierigkeiten bei der Bestimmung der mathematischen Eigenschaften mancher abstrakter Größen ergeben können. Es lohnt sich jedoch selten, eine vollständige, detaillierte Analyse vorzunehmen, so dass wir immer bestrebt sind abzuschätzen, um auf Einzelheiten verzichten zu können. (Tatsächlich erweisen sich grob erscheinende Schätzungen oft als recht genau.) Solche groben Schätzungen lassen sich sehr oft leicht aus der alten Faustregel ableiten, welche da lautet: >>90 % der Zeit werden für

[68] *PS2*: Playstation 2, Hersteller ist Sony

[69] *Xbox*: Hersteller ist Microsoft

10 % des Programms benötigt.<< (In der Vergangenheit wurde diese Aussage für viele von 90 % verschiedene Werte formuliert.)“ [70]

Robert Sedgewick

Aufgrund der Tatsache, dass die Entwicklung von Engines meistens einem sehr straffen Zeitplan folgt, muss abgeschätzt werden, wieweit eine Optimierung umsetzbar ist. Der A* gehört allerdings zu den von Robert Sedgewick angesprochenen 10% der Algorithmen, die eine Grundoptimierung benötigen.

Aufgrund seiner Grundstruktur und Funktionsweise ist der A* ein langsamer Algorithmus. Ungünstigerweise muss der A*, weil in den meisten Spielen Charaktere bewegt werden, sehr häufig verwendet werden. Die Auswirkungen für die Performanz eines unoptimierten Pathfindingsystems ist verheerend. Es gibt eine Vielzahl von Möglichkeiten, den A* zu optimieren.

Der Hauptgrund für die Langsamkeit des A* ist in der Regel der enorme Suchbereich. Bei einer regular grid von 1000 mal 1000 Quadraten beträgt der Suchbereich beispielweise eine Millionen Quadrate. Unabhängig von der Optimierung des Quellkodes bedarf die Suche nach dem kürzesten Weg eines ungeheuren Aufwandes. Dies hat zur Folge, dass die primär durchzuführende Optimierung sich diesem Bereich widmet. Erst nachdem der Suchbereich eingeschränkt wurde, ist es ratsam, sich mit der Implementierung zu beschäftigen. Im Bereich der Implementierung muss die Alloziierung[71] des Speicherplatzes und der Zugriff auf die Datentypen optimiert werden. Dem extremen Suchaufwand begegnet man mit speziellen, schnellen, effizienten, auf die Suche ausgerichtete Datentypen. Die beste Art den A* zu optimieren ist, ihn nur dann zu verwenden, wenn es auch notwendig ist. Aus diesem Grund sollte ein System entwickelt werden, dass erkennt, ob eine Notwendigkeit überhaupt besteht. Eine Aufgabe des Systems ist beispielweise zu ermitteln, ob es eine prinzipielle Möglichkeit gibt, zwischen zwei Punkten einen Pfad

[70] Zitat: Sedgewick, Algorithmen in C++, S.95

[71] *Alloziierung:* Speicherbereitstellung

zu erstellen. Des weiteren kann ein schneller Test durchgeführt werden, ob es einen direkten Pfad zwischen den Punkten gibt.

Suchbereichsoptimierung - Erstellung der minimalen Suchmenge

Da der Suchaufwand proportional zur Suchmenge steigt, ist die primäre Optimierung zur Beschleunigung des A* die Verkleinerung des Suchbereichs. Aus diesem Grund wird bei der *hierarchischen Suche* zuerst ein allgemeiner Pfad berechnet. Der kleine Suchbereich auf den für die Suche des allgemein Pfades zugegriffen wird, umfasst erst einmal nur die Verbindungspunkte zwischen den Lokations oder Areas. Anhand dieses kleinen Suchbereichs kann beispielsweise schnell herausgefunden werden, ob es überhaupt einen Weg von A nach B gibt.

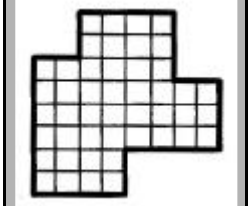
Die Möglichkeiten Suchbereiche, z.B. anhand rectangular grids oder Polygonen, verkleinert abzubilden, wurde bereits in einem vorherigen Kapitel erwähnt. Nun sollen diese Möglichkeiten etwas genauer untersucht und erläutert werden, um zu zeigen, wie spezielle Suchbereiche anhand spezieller Suchmengen abzubilden sind.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Rectangular/ Hexagonal Grid – Quadratisches/ Sechseckiges Netz

Bei dieser Suchbereichsminimierung wird die Welt[72] oder Lokation als Netz, bestehend aus gleichgroßen Quadraten oder Sechsecken, dargestellt. Die Größe der Quadrate oder Sechsecke wird durch die Größe des kleinsten Hindernisses oder des kleinsten Charakters bestimmt. Es ist allerdings auch möglich den Quadraten eine feste Größe von beispielsweise 1, 4, 16, etc. Pixel zuzuweisen. Befindet sich ein Objekt in einem Quadrat, dann wird diese Stelle nicht in die Suche einbezogen.

Abbildung 21: Rectangular Grid



Der Vorteil dieser Methode ist, dass sehr schnell herausgefunden werden kann, ob ein Bereich begehbar ist. Die Überprüfung eines Bereichs bedarf unter gewissen Umständen eines Zugriffes von nur O(1). Eine hervorragende Eigenschaft dieser Methode ist, dass nicht nur Bereiche mit festen Objekten, sondern auch solche mobiler NPCs betrachtet werden können. Der Nachteil besteht darin, dass die Suchmenge immer noch relativ groß ist. Des weiteren ist es schwierig Grids für die Abbildung von 3D Welten zu verwenden. Meistens verwendet man sie nur in der 2D Spieleprogrammierung. Die durch die generierten Pfade entstehende Bewegung tendiert dazu schachbrettartig zu erscheinen. Um dies zu vermeiden, sollten die einzelnen Quadrate nicht zu groß gewählt werden.

[72] *Welt*: die Welt ist die Gesamtheit aller Level

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Actual Polygonal Floor – tatsächliche polygonale Abbildung des Fußbodens

Aufgrund der Tatsache, dass in 3D Welten alle Objekte anhand von Polygonen geometrisch gerendert werden und es eine Möglichkeit gibt die Polygone, die den Boden darstellen, entsprechend zu markieren, kann diese Darstellung direkt als Suchbereich verwendet werden.

Abbildung 22: Actual polygonal floor

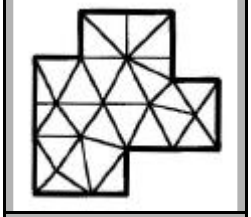
Der Vorteil dieser Methode ist, dass die Daten im 3D Bereich schon in dieser Form vorliegen (im 2D Bereich können sie allerdings auch erstellt werden, siehe C-cells) und schnell mit BSP Trees durchsucht werden können. Der Nachteil ist, dass gegebenenfalls die Böden anhand unheimlich vieler Polygone dargestellt werden. Hindernisse, die oberhalb des Bodens liegen, wie z.B. Tische und Stühle, werden bei dieser Methode nicht in Betracht gezogen. Des weiteren bedarf es eines Algorithmus, der innerhalb der Polygone bewertet, welche Punkte begehbar sind.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Polygonal Floor Representation – spezielle polygonale Abbild des Fußbodens

Bei dieser Methode wird nicht die sowieso existierende Welt in Polygonen als Suchmenge verwendet, sondern eine zweite polygonale Abbildung der Welt, die nur die begehbaren Bereiche umfasst und separat erstellt wird.

Abbildung 23: Polygonal floor representation



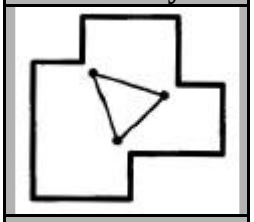
Der Vorteil dieser Suchbereichsabbildung ist, dass der Suchbereich sehr klein ist. Des weiteren kann auch bei dieser Technik der Bereich mit BSP Trees durchsucht werden. Die Hindernisse sind bereits mit in die Map eingearbeitet. Der Nachteil ist, dass zusätzliches Personal gebraucht wird, um die Maps zu erstellen. Außerdem können keine mobilen Objekte, wie beispielsweise Charaktere, abgebildet werden. Ein weiterer Nachteil ist, dass auch hierbei ein Algorithmus verwendet werden muss, der innerhalb der Polygone bewertet, welche Punkte begehbar sind.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Points of Visibility – Sichtbarkeitspunkte

Bei dieser Methode werden um alle Hindernisse Punkte gelegt. Diese Punkte werden mit allen adäquaten Punkten, die sich in der Nähe des Punktes befinden, verbunden, um einen Laufgraphen zu entwickeln. Mittels dieses Graphen ist der minimale Pfad, der um ein Hindernis führt, bekannt.

Abbildung 24: Points of visibility



Der Vorteil dieser Methode ist, dass der Suchbereich minimal ist, Hindernisse schon mit berücksichtigt werden und die entstehenden Pfade hinsichtlich der Pfadsuche schon perfekt sind. Der Nachteil ist, dass ein Algorithmus implementiert werden muss, der einen solchen Graphen entwirft. Es besteht aber auch die Möglichkeit den Graphen von einem Level Designer erstellen zu lassen. Ein weiterer Nachteil ist, dass Hindernisse nicht dynamisch entfernt oder hinzugefügt werden können. Charaktere werden bei dieser Methode nicht berücksichtigt. Bei Entitäten mit besonders großer Breite funktioniert diese Methode nicht sonderlich gut. Bei der Abbildung von Welten mit runden Wandverläufen kann der Graph sehr komplex werden.

Fazit

Es ist leicht erkennbar, dass man die Wahl der Welt- bzw. Lokationabbildung abhängig von der verwendeten Welt bzw. Lokation treffen muss. Man sollte beachten, dass die Art der Suchbereichsverarbeitung dem Speicherbedarf angepasst werden muss. Kann die Engine beispielsweise viel Speicher für die Pfadfindung zur Verfügung stellen, dann ist es möglich, dass die Map, die für die Suchmenge erstellt wird, speichertechnisch größer ist. Dies hat zur Folge, dass die Suchdaten

gegebenenfalls schneller ermittelt werden können und die Suchmenge dennoch klein ist.

Optimierung des Algorithmus

Nachdem der Suchbereich minimiert wurde, kann man sich der Verbesserung der A* Implementierung widmen.

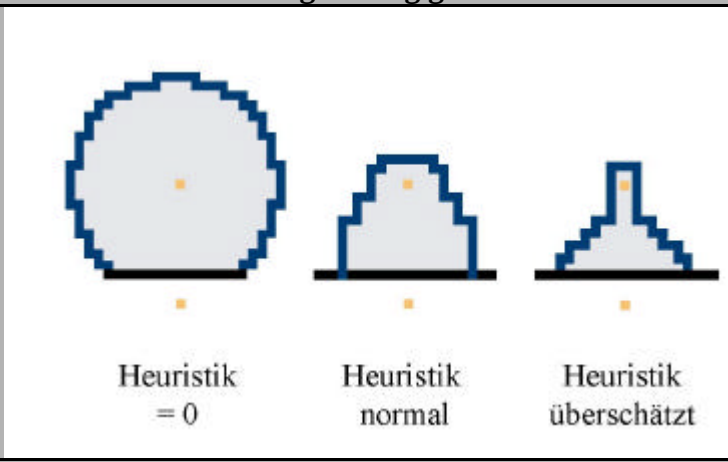
Die Heuristik

Normalerweise wählt man bei der Implementierung des A*- Algorithmus eine heuristische Distanz, hierbei handelt es sich um die abgeschätzte Distanz zwischen dem Such- und dem Zielknoten, die kleiner ist, als die tatsächliche Distanz zwischen den beiden Punkten. Bei dieser Vorgehensweise wird die tatsächliche Distanz unterschätzt. Die Heuristik ist in der Regel die Luftlinie zwischen den Punkten, die mit Hilfe des Satz des Pythagoras errechnet wird. Diese Methode garantiert, wenn es sich um zwei legale Punkte der Suchmenge handelt, dass definitiv der kürzeste Weg ermittelt wird. Wäre die Möglichkeit gegeben, dass die heuristische Distanz tatsächlich immer gleich der tatsächlichen Distanz ist, dann wäre die Ermittlung des kürzesten Weges sehr schnell. In der Regel ist dies nicht der Fall. Es gibt allerdings eine Möglichkeit, auf die bereits hingewiesen wurde, die Geschwindigkeit anhand der Verwendung einer anderen heuristischen Distanz zu beschleunigen. Der Geschwindigkeitsgewinn revidiert allerdings den Anspruch an den perfekten kürzesten Pfad. Es entsteht aber immer noch ein sehr kurzer Weg und dies in kürzerer Suchzeit. Der Weg zu einer Geschwindigkeitsoptimierung führt über eine Überschätzung der heuristischen Distanz.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Die Überschätzung der heuristischen Distanz führt dazu, dass die Suche in Richtung Zielpunkt noch weiter forciert wird. Verwendet man eine heuristische Distanz von 0, dann breitet sich die A* Suche kreisförmig aus. Wählt man die Heuristik im Sinne des Satz des Pythagoras, dann breitet sich die Suche oval aus. Wählt man nun eine überschätzte Heuristik, dann folgt die Suche einer Diamantform.

Abbildung 25:
Suchausbreitung abhängig von der Heuristik



Mit Hilfe dieser Überschätzung der heuristischen Distanz findet bei der Suche eines Pfades, bei der sich zwischen dem Start- und Zielpunkt größtenteils „gerade“ Hindernisse befinden, eine Beschleunigung der Suche statt. Eine solche Einschränkung weist schon darauf hin, dass eine Überschätzung nicht immer vorteilhaft ist. Prinzipiell wird die Suche mittels der Überschätzung in Richtung des Zielpunktes sehr stark forciert. Dies führt zu dem Umstand, dass die Suche in dem Fall verlangsamt wird, in dem sie sich von einem Punkt zurück in Richtung Startpunkt bewegt. Somit sollte die Suche mittels der Überschätzung in den Bereichen vermieden werden, in denen extreme Richtungswechsel stattfinden.

Zur besseren Veranschaulichung betrachte man noch einmal das Beispiel der Suche mit konkaven Hindernissen.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Abbildung 26:
Konkaves Hindernis

Wird das Hindernis erreicht, dann folgt daraus, weil das Hindernis konkav ist, eine Bewegung in Richtung Start-, weg vom Zielpunkt. In diesem Fall sollte man, anstatt der überschätzen Heuristik, die „normale“ Heuristik (anhand der Luftlinie) verwenden.

Prinzipiell bedeutet dies, dass eine Beschleunigung anhand der überschätzten heuristischen Distanz möglich und sehr effektiv ist, dass aber eine Analyse der Welt notwendig ist. Diese Methode kann also in den Fällen verwendet werden, in denen eine solche Analyse den Geschwindigkeitserfolg nicht mindert und in denen ein beinahe kürzester Weg ausreicht. Da in der Regel die Welten ohnehin konstruiert werden, kann eine Analyse auch durch das Geschick von Level Designern[73] ersetzt werden.

Prinzipiell bedeutet dies, dass mit der Heuristik experimentiert werden muss. Gute Indikatoren, um zu überwachen, wie effektiv der A* arbeitet, sind die Größe der Closed List und die maximale Größe der Open List. Die maximale Größe der Closed List gibt Aufschluss darüber, wie viele Knoten getestet wurden. Die maximale Anzahl der Open List ist der Indikator, wie viel Zeit aufgewendet wurde, um einen Knoten zu kontrollieren. Zur Ermittlung der besten und effektivsten heuristischen Distanz bewertet man die Zeit, die aufgebracht werden muss, um eine Suche mittels der

[73] *Level Designern*: Personen, die anhand von Editoren und Graphiken ein Level oder eine Lokation zusammenbauen.

Luftlinie durchzuführen, hinsichtlich der Zeiten, die anhand der überschätzen heuristischen Werte entstehen.

Um zu vermeiden, dass der zu generierende Pfad zu sehr vom kürzesten Pfad abweicht, muss auch in dieser Richtung experimentiert werden.

Speicheroptimierung - Entkopplung der Pfadfindungsdaten von der Suchmenge

Um die A* Suche durchzuführen bedarf es viel Speichers, um alle relevanten Information zur Verfügung zu haben. Normalerweise koppelt man diesen Speicherplatz an die Knoten selbst. Bei einer regular grid beinhaltet somit jedes Quadrat, bei der Polygonalsuche jedes Dreieck, Pfadfindungsdaten. Da es in der Regel nicht vorkommt, dass jeder Knoten untersucht wird, wird bei der Kopplungsmethode viel Speicher verschwendet. Bei einer 1000 * 1000 Quadratmap bedarf man eines Speicherplatzes für 1 Millionen mit Informationen vollgestopfter Pfadfindungsknoten. Die Lösung besteht darin, den Speicher der Pfadfindungsdaten von der Menge aller möglichen Laufpunkte, der Suchmenge, zu trennen. Dies hat eine außerordentliche Speicherentlastung und einen Geschwindigkeitszuwachs der Suche zur Folge.

Bei der Entkopplung der Pathfindingdaten von der Suchmenge wird nunmehr ein Speicherblock alloziiert, der für die einzelnen Knoten der tatsächlichen Suchmenge zur Verfügung gestellt wird, d.h., erst wenn ein Knoten untersucht wird, erhält dieser einen Teil des alloziierten Speicherblocks. Beim A* sollte darauf geachtet werden, dass so wenig dynamischer Speicher wie möglich verwendet wird, weil die Bereitstellung und Freigabe von Speicher mit *new* und *delete* eine sehr zeitaufwendige Prozedur ist. Somit sollte darauf geachtet werden, dass für den A* nur einmal Speicher zur Verfügung gestellt wird.

Es muss natürlich getestet werden, wie groß dieser Speicher sein muss, um zu vermeiden, dass durch zu wenig Speicher das Programm abstürzt. Im Notfall sollte es deshalb doch eine Möglichkeit geben, den Speicher dynamisch zu erweitern.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Es stellt sich die Frage wofür der A* so viel Speicher braucht. Informationen, die für jeden Knoten gespeichert werden müssen sind:

1. Zeiger zum Elternknoten.
2. Distanz Startknoten zum Knoten.
3. ganze Strecke (Distanz Startknoten zum Knoten + heuristische Abschätzung zum Zielknoten).
4. Variabel, ob sich der Knoten in der open oder closed list befindet (oder zwei Variabeln).

Außerdem muss entschieden werden, welcher Variabeltyp verwendet werden soll: unsigned char, int, etc. Signed oder unsigned int sollte man nicht verwenden, weil laut Microsoft Dokumentation die Größe für int abhängig vom System variiert.

Master Node List

Den alloziierten Speicherblock der Knoten nennt man auch eine Knotenbank. Sobald ein Knoten aus der Knotenbank für die Suche verwendet wird, muss er irgendwo gespeichert werden, um schnell auf ihn zugreifen zu können. Für diese Art der Speicherung ist eine Hash Table die optimale Datenstruktur. Mit Hilfe der Hash Table hat man die Möglichkeit mit einem Zugriff von bestenfalls O(1) auf die Informationen des Knotens zuzugreifen. Da sich die Knoten speichertechnisch in der Knotenbank befinden, müssen nur Zeiger auf die Knoten in der Hash Table gespeichert werden.

Die Master Node Hash Table verwaltet sowohl alle Knoten der Open, als auch der Closed List. Eine Variable gibt Auskunft darüber, in welcher virtuellen Liste sich ein Knoten befindet. Es ist nicht nötig die beiden Listen zu trennen. Die Open List wird außerdem noch in einer weiteren Liste geführt. Der Grund dafür ist, dass der Zugriff auf die Werte der Knoten unter gewissen Umständen in der eigentlichen Open List, wie man sehen wird handelt es sich hierbei um eine priority queue, unter anderen, in der Master Node List schneller ist.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Um zu kontrollieren, ob ein Knoten schon verwendet wird, sollte es eine Funktion geben, die diese Aufgabe übernimmt. Wenn sich der Knoten schon in der Master List befindet, wird dieser zurückgegeben, ansonsten ein freier Knoten aus der Knotenbank, der allerdings noch zu initialisieren ist.

Optimierung der Open List

Das Faszinierende des A*- Algorithmus ist, dass die Suche hinsichtlich der vielversprechendsten Suchrichtung gelenkt wird. Dies funktioniert, indem die potentiellen, als nächstes zu kontrollierenden Knoten der Open List hinzugefügt und entsprechend des vielversprechendsten Knoten sortiert werden. Das Problem ist, dass die Open List dahin tendiert sehr groß zu werden, was auch den Sortieraufwand negativ beeinflusst. Der geeignetste Datentyp, um dieser Problematik entgegen zu wirken, ist die *Prioritätsschlange*[74]. Die verwendete priority queue sollte als binary heap implementiert werden. Eine binary heap ist ein sortierter Baum, welcher die Eigenschaft hat, dass die Elternknoten einen geringeren Wert (Ebene) besitzen, als die Kindknoten. Dieser Baum ist nicht vollständig sortiert; es findet keine direkte Sortierung unter den Schwesternknoten statt. Es muss nur dafür gesorgt werden, dass alle Werte einer Ebene, wenn nach dem geringsten Wert sortiert wird, kleiner als die Werte der „darunter liegenden“ Ebene sind, d.h. die Kindknoten sind größer als die Elternknoten und deren Schwesternknoten. Aufgrund dieser hervorragenden Eigenschaft braucht man zum Hinzufügen irgendeines Knotens und Entfernen des vielversprechendsten Knotens einen Aufwand von nur O(log n). Auch bei der PQ werden nur Zeiger gespeichert, weil sich die Knoten speichertechnisch in der Knotenbank befinden.

[74] *Prioritätsschlange*: priority queue

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Implementierung einer Prioritätsschlange

Die A* Open List Prioritätsschlange bedarf vier Methoden mit gewissen Eigenschaften:

1. Entferne den Knoten mit der geringsten Gesamtdistanz: O(log n).
2. Füge neuen Knoten zur Queue hinzu und sortiere die Queue neu: O(log n).
3. Führe ein Update der Gesamtkosten eines bestimmten Knotens durch und sortiere Queue neu: O(n + log n).
4. Besage, ob die Open List leer ist: O(1).

Die Stärke der PQ liegt in der Ermittlung des vielversprechendsten Knotens und die der Master Node List in der allgemeinen Bereitstellung und Ermittlung der Knoten.

Fazit

Aufgrund der Tatsache, dass der A* ein äußerst rechenintensiver Algorithmus ist, sollte ,bevor Optimierungen durchgeführt werden, ein Abbild der Welt bzw. der Lokation in einfachster Form vorliegen. Man sollte die Level Designer auf die Grundproblematiken der A* Suche aufmerksam machen, damit schon hier kritische Fälle abgefangen werden können. Bevor man sich der Optimierung widmet, sollte der A* fehlerfrei funktionieren, denn das Debugging ist sehr kompliziert. Wenn man mit der Optimierung beginnt, sollte zuerst die Entkopplung vom Suchbereich durchgeführt werden. Danach sollte man eine Prioritätsschlange für die Open List und die Hash Table für die Open und Closed List implementieren. Erst zum Schluss sollte man beginnen, mit der Heuristik zu experimentieren. Man wird feststellen, dass man die Heuristik im Laufe der Zeit, wenn die Repräsentation der Welt bzw. der Lokation besser verständlich wird, noch oft verbessern wird. Die ästhetische Optimierung sollte erst nach der Performanz- und Speicheroptimierung durchgeführt werden.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

7.0

Praktische Umsetzung des Pathfindings

Voraussetzungen

Entscheidungen

Bevor man mit der Umsetzung einer AI Engine, in diesem Falle mit einem Pathfindingsystem beginnt, muss man das Spielgenre betrachten, d.h. eine Analyse im Hinblick auf das Endziel durchführen.

Im Falle des Beispielprogramms wird eine 2D Engine verwendet. Diese Engine ist nicht konsolenkompatibel, d.h. dass ein extremes Speicherproblem nicht auftreten wird. Allerdings ist der Speicherverbrauch hinsichtlich der Graphikengine relativ groß. Es ist der Anspruch des Pathfindingsystems so wenig Speicher wie nötig zu verwenden und die Suche so schnell wie möglich zu gestalten. Aus diesem Grund wird es kein LOD[75] AI System geben, weil immer der beste Pfad in kürzester Zeit berechnet werden soll.

Prinzipiell sollen drei *phänomenale Gestalten* implementiert werden. Die Lokation ist ein schneebedeckter Marktplatz mit normalen Wegen, Brücken und Eis. Alle drei Spielfiguren sollen sich entsprechend ihres Charakters auf diesem Marktplatz bewegen, unter der Vorraussetzung, dass sie versuchen, den kürzesten Weg von einem

[75] *LOD*: Level Of Detail, hierbei handelt es sich um die Möglichkeit auf unterschiedliche AI Situationen unterschiedlich zu reagieren. Im Fall des Pathfindings bedeutet dies, dass das Pathfinding des PCs ästhetischer umgesetzt wird, als beispielsweise das eines NPCs, der einmalig im Spiel auftaucht.

Punkt A nach B zu generieren. Der Charakter Gabriel sucht immer den kürzesten Weg. Für ihn spielt es keine Rolle, ob er über Schnee, über die Brücken oder über Eis gehen soll. Ein kleiner Junge namens Avron liebt nichts mehr als Eis. Aus diesem Grund entscheidet er sich, wenn bei seiner Suche an eine Area eine Area mit Eis andockt, immer für den Weg über das Eis, auch wenn ein Weg über eine andere Area kürzer ist. Der dritte Charakter, Stuart, hat Angst, dass er auf dem Eis einbrechen könnte. Aus diesem Grund wird er niemals eine Eisarea betreten. Prinzipiell gibt es drei Eisareas. Das Demoprogramm soll zeigen, dass die Suche nach einem kürzesten Weg an eine Motivation gekoppelt werden kann. Aufgrund dieses Ziels hat, um das Laufen der Figuren so realistisch wie möglich erscheinen zu lassen, die ästhetische Umsetzung mehr Priorität, als die Performanz- und Speicheroptimierungen, d.h. Schönheit vor Schnelligkeit und Speicher.

Spezifikation

Allgemein

1. Ermittlung des kürzesten Pfades mit sehr guter ästhetischer, speicher- und performanzspezifischer Optimierung, wobei die Ästhetik aufgrund des Anspruches an die *phänomenale Gestalt* am meisten gewichtet wird.
2. Pathfinding abhängig von einer Entscheidungsfindung.

Pathfindingsystem

1. Durchführung einer Analyse zur Ermittlung eines direkten Weges ohne A*
2. kürzeste Wegberechnung mit dem A*

Motivation

1. terrainabhängige Entscheidungsfindung

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Allgemein

Einleitung

„Designing an algorithm for the heuristic cost can at times seem more like voodoo than science.“[76]

Steve Rabin

Die heuristische Abschätzung des A* ist der Grund für die schnelle Suchleistung dieses Algorithmus. Die Heuristik beeinflusst die Suchrichtung dermaßen stark, dass man äußerst eigenwillige Methoden entwickeln muss, um den A* von seiner Arbeitsweise abzubringen. Mitunter ist es notwendig auf die Suche Einfluss zu nehmen, weil ihr Ergebnis nicht befriedigend ist. Diese Einflussnahme auf den A* ist dass, was Steve Rabin als Voodoo bezeichnet.

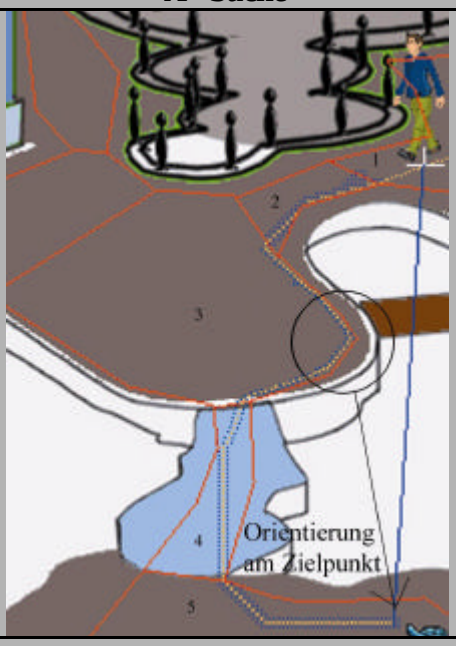
Prinzipiell gibt es aufgrund des Einflusses der Heuristik auf die Suchrichtung keinen besseren Algorithmus als den A*, um schnell den kürzesten Weg zwischen zwei Punkten zu generieren. Der Grund dafür ist, dass der Algorithmus die Suchmenge sehr stark eingrenzt, weil es sich um eine gerichtete Suche handelt. Bei der Dijkstra Suche breitet sich die Suche um den Startpunkt kreisförmig aus, sodass eine sehr große Suchmenge von kontrollierten Punkten entsteht. Beim A* wird nur der vielversprechendste Punkt betrachtet. Anhand dieses Punktes breitet sich die Suche in Richtung des Zielpunktes aus, sodass, im Gegensatz zum Dijkstra, nur ein Bruchteil aller möglichen Punkte kontrolliert wird.

[76] Zitat: Steve Rabin, Game Programming Gems, S.276

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Um eine *phänomenale Gestalt* zu entwickeln, bedarf es einer Weiterentwicklung der A* Suche, die dem *hierarchischen Pathfinding* ähnelt, weil der A* nicht dazu geeignet ist, einen natürlichen Pfad zu generieren, der glaubhaft vermittelt, dass ein Mensch ihn so gehen würde. Das Problem ist, dass der Pfad dermaßen stark in Richtung des Zielknotens tendiert, dass der Charakter sehr kurvenmäßig läuft.

Abbildung 27:
Unästhetisches Ergebnis der normalen A* Suche



Bei dem für die Diplomarbeit entwickelten System spielt wieder einmal die Einflussnahme auf die Heuristik die entscheidende Rolle.

Ein Vorteil der neu entwickelten Methode ist, neben der Generierung eines sehr ästhetischen Pfades, dass eine *smooth* und *straight path* Methode, aufgrund der Kombination mit der gewählten regular grid, nicht mehr notwendig ist, weil der Pfad automatisch natürliche Kurven entwickelt und geglättet wird.

Um ein gutes Pathfindingsystem zu implementieren muss man sich zuerst einmal von der Vorstellung lösen, dass der A* selbst ein Pfadfindungssystem darstellt.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Der A* ist nur der Algorithmus, der schnellstmöglich einen perfekten Pfad ermittelt, aber es bedarf, wie auf der oberen Abbildung zu sehen ist, weitaus mehr, um ästhetische Pfade zu generieren.

Die für die Demoversion verwendete 2D Engine generierte bisweilen den Laufpfad für die Charaktere anhand des Dijkstra- Algorithmus. Aufgeteilt sind die Lokations in sogenannte *Laufareas* und *–pfade*, die bestimmen, wo in der Lokation gelaufen werden darf. Ein *Pfad* ist eine Linie zwischen zwei Punkten auf der gegangen werden darf. Eine *Area* ist ein Vieleck, bestehend aus mindestens vier und maximal neunzehn Eckpunkten, das ebenfalls einen betretbaren Bereich eingrenzt. Die Außenlinien, die zwei benachbarte Areas gemein haben, über die sie praktisch verbunden sind, nennt man *Portale*. Das bisherige Pfadfindungssystem generierte Pfade von einer Area zur nächsten, wobei von den Portalen nur die Eckpunkte verwendet werden durften. Ansonsten waren alle Punkte der Areas und Pfade zulässig. Zwischen zwei Pfaden oder einem Pfad und einer Area gibt es nur einen Portal-/ Verbindungspunkt.

Agent

Um ein sehr flexibles System zu generieren, sollten, wenn die Entscheidung getroffen wird ein vollkommen neues Pfadfindungssystem für eine Engine zu entwickeln, alle Feinheiten beachtet werden. Aus diesem Grund wurde beschlossen, den vorhandenen Charakteren, die aus einer Struktur (die Engine ist größtenteils in C programmiert) 'Tactor' bestehen, um die Komponente Agent (class GkCAgent) zu erweitern.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Definition: Ein *Agent* ist eine physische oder virtuelle Entität

1. die selbstständig in einer Umwelt agieren kann,
2. die direkt mit anderen Agenten kommunizieren kann,
3. die durch eine Menge von Absichten angetrieben wird (in Form von individuellen Zielen, Befriedigungs- und Überlebensfunktionen, die sie versucht zu optimieren),
4. die eigene Ressourcen besitzt,
5. die fähig ist, ihre Umwelt wahrzunehmen (allerdings nur in bestimmtem Ausmaß),
6. die nur eine partielle Repräsentation ihrer Umwelt besitzt,
7. die bestimmte Fähigkeiten besitzt und Dienste offerieren kann,
8. die sich ggf. selbst repräsentieren kann,
9. deren Verhalten darauf ausgerichtet ist, ihre Ziele, unter Berücksichtigung der ihr zur Verfügung stehenden Ressourcen und Fähigkeiten, zu befriedigen und die dabei auf ihre Wahrnehmung, ihre internen Modelle und ihrer Kommunikation mit anderen Agenten (oder den Menschen) angewiesen ist.

Man hätte ’Tactor’ nur um die Komponente Pfadfindungssystem erweitern können. Da sich die Diplomarbeit aber auf eine *phänomenale Gestalt* bezieht, die mehr als nur laufen kann, wurde es für notwendig erachtet, auf einen *Agenten* zu bestehen.

Die Charaktere folgen auch dieser Definition eines *Agenten*. Gabriel, Avron und Stuart handeln sehr individuell. Sind die Charaktere NPCs, dann agieren sie selbstständig in ihrer Umwelt, sie laufen nach einem Zufallsprinzip umher. Eine direkte Kommunikation findet nicht statt, wäre allerdings möglich. Alle drei verfolgen Ziele, sowie Befriedigungs- und Überlebensfunktionen. Gabriel hat das Ziel immer den kürzesten Weg zu finden. Avron liebt nichts mehr, als über Eis zu laufen und Stuart läuft niemals über Eis. Sie sind in der Lage ihre Umwelt durch eine partielle Repräsentation wahrzunehmen. Die Repräsentation ihrer Umwelt ist eine regular grid (const GkCLocationGrid *m_WorldKnowledge). Den Charakteren ist aufgrund dieser Grid bewusst, dass sie nur über den Schnee, die Brücken und das Eis laufen können. Alle Charaktere besitzen die Fähigkeit zu laufen und zu entscheiden, wo sie laufen

dürfen. Ihr höchstes Ziel ist es das Laufen in ihrem Sinne so perfekt wie möglich abzubilden.

In diesem Sinne sieht die Agentenklasse folgendermaßen aus:

<source code>

```
class GkCAgent
{
        public: // Kon-/ Destruktor
        GkCAgent(GkInt16 Int16ID, GkFloat FloatXPosition, GkFloat FloatYPosition);
        ~GkCAgent();
public:
        // returns xPosition of entity
        const GkFloat GetXPosition() const;
        // returns yPosition of entity
        const GkFloat GetYPosition() const;
        // returns identifier of entity
        const GkInt16 GetIdentification() const;
public:
        // A* Pathfinding method
        GkInt08 FindPath(const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition);

        // Setup of World Knowledge
        void SetupWorldKnowledge(const GkCLocationGrid *m_WorldKnowledge, GkCVector<GkCPathfindingNode>
                *gkCNodeBankVector, const GkCPortalManager *gkCPortalBank, const GkSSpecialAreas *gkSNoAreas =
                NULL, const GkSSpecialAreas *gkSPreferedAreas = NULL);
        // sets heuristic type
        void SetHeuristicType(const GkuInt08 ui08HeuristicType);
        // Get Path destination
        const GkPoint& GetDestination() const;
        // return current movement point and increases iterator for  used points
        const GkSMovInfo* GetNextWayInfo() const;
        // return current movement point without increasing iterator for  used points
        const GkSMovInfo* CheckNextWayInfo() const;
public:  // debug infos
        // returns amount of Nodes of the path
        const GkuInt32 GetNodes() const;
        // returns time needed to create the path
        const GkDouble GetSearchTime() const;
        // DrawPath
        void DrawPath(GkuInt08 ui08Path);

protected:
        GkCPathfindingSystem *m_GkCPathfindingSystem;   // Pathfinding system
private:
```

```
        GkFloat m_FloatXPosition;        // x - Postion of the entity in its world
        GkFloat m_FloatYPosition;        // y - Postion of the entity in its world
        GkInt16 m_Int16ID;               // identifier of the agent
};
```

</source code>

Für das Pathfinding sind eigentlich nur zwei Methoden interessant:

- GkInt08 FindPath(const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition);
- void SetupWorldKnowledge(const GkCLocationGrid *m_WorldKnowledge, GkCVector<GkCPathfindingNode> *gkCNodeBankVector, const GkCPortalManager *gkCPortalBank, const GkSSpecialAreas *gkSNoAreas = NULL, const GkSSpecialAreas *gkSPreferedAreas = NULL);

FindPath(..) dient dazu den Pfad zu generieren. Es muss nur der Zielpunkt übergeben werden, weil der Startpunkt die Position des *Agenten* (GkCAgent::m_FloatXPosition, GkCAgent::m_FloatYPosition) ist. SetupWorldKnowledge(..) muss zuvor einmal aufgerufen werden, um dem *Agenten* das Wissen über die Lokation, also die Lokationgrid, zu übermitteln und des weiteren die Knotenbank, aus der sich der Agent bedienen kann, um Speicher für seine Suche zu erhalten. Für die Motivation können hier des weiteren die Areanummern übergeben werden, die ein bestimmter *Agent* nicht betreten darf (gkSNoAreas) und die, die er bevorzugen soll (gkSPreferedAreas).

Prinzipiell hat der *Agent* eine Member- Variable GkCPathfindingSystem *m_GkCPathfindingSystem, die das Pathfindingsystem repräsentiert und die Pfade generiert.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Die phänomenale Gestalt – Teil 1

Um eine *phänomenale Gestalt* zu erstellen, bedarf es der Umsetzung der drei Grundmechanismen der Charakter AI. Es muss das Lokomotion, das Steering und die Motivation umgesetzt werden.

Lokomotion & Steering

Lokomotion

Beim Lokomotion geht es darum, den Charakter so zu stellen, dass es aussieht, als ob er von der Körperstellung her in die richtige Richtung läuft. Der Charakter darf z.B. nicht rückwärts stehen, wenn er vorwärts läuft. Aus diesem Grund sollte permanent geprüft werden, ob der Pfad des Charakters die Richtung wechselt, weil der Charakter dann auch einer anderen Laufanimation bedarf. Prinzipiell hat der Charakter acht Laufanimationen/ - richtungen: Er kann

- nach oben,
- nach links oben,
- nach links,
- nach links unten,
- nach unten,
- nach rechts unten,
- nach rechts,
- und nach rechts oben laufen.

Aufgrund der Tatsache, dass der Charakter nur acht Laufrichtungen hat und die Änderung der Laufrichtung performanztechnisch sehr aufwendig ist, wirkt diese mitunter sehr ruckelig. Prinzipiell wird permanent die vorberechnete Steigung zwischen dem Laufpunkt und seinem Vorgänger geprüft und anhand dieses Wertes ggf. die Laufanimation geändert. Da die Performanz, wie man sich vorstellen kann,

sehr unter dieser Methode gelitten hat, wurde beschlossen, die Steigung nur bei jedem fünften Punkt zu kontrollieren. Diese Maßnahme reduziert den Ruckeleffekt sehr stark und liefert immer noch ein gutes ästhetisches Ergebnis. Eine solche Vorgehensweise hat allerdings zur Folge, dass unter extremen Voraussetzungen, d.h. wenn sich die Steigung um ca. 90 Grad ändert, der Charakter ggf. einen kurzen Moment falsch steht. Diese Tatsache musste aber ignoriert werden, weil ein Umbau der Engine hinsichtlich des Ladens von Laufanimationen Wochen gedauert hätte, vielleicht sogar gar nicht möglich gewesen wäre. Zum schnelleren Testen wurde des weiteren eine „Vorspul“ – Möglichkeit eingebaut, die es erlaubt, dass die Charaktere dreimal so schnell laufen (Drücken der F – Taste). Hierbei fällt schon eher auf, wenn der Charakter nicht in der richtigen Richtung steht.

Des weiteren müsste sich die Größe der Schritte, die die Charaktere gehen, verändern, wenn die Größe der Charaktere skaliert wird. Aufgrund der Tatsache, dass es sich bei dem Programm um eine Demoversion handelt und die Lokomotion nicht das Hauptthema der Diplomarbeit ist, wurde dieser Aspekt vernachlässigt.

Steering

Die Lösung des Pathfindingproblems ist eines der größten und schwierigsten Unterfangen der gesamten AI. Das Problem ist, dass ein Pathfindingkonzept eine allgemeine Lösung darstellt, Lokations aber den Inbegriff der Individualität verkörpern. Keine Lokation gleicht einer anderen, alle Hindernisse sehen verschieden aus, haben unterschiedlich viele Eckpunkte und ihre Größen sind niemals gleich. Ein Pathfindingkonzept sollte dementsprechend viele Lokation- und Hindernisarten bewerkstelligen können, es ist allerdings im Sinne der Performanz, den Leveldesignern gewisse Areakonstrukte vorzuschreiben.

Das größte Problem des Pathfindings ist, dass es sehr oft so dargestellt wird, als ob, wie erwähnt, der A* ein Pathfindingkonzept sei, mit dessen Hilfe alle Probleme des Pathfindings gelöst werden könnten. Aus diesem Grund sollte darauf hingewiesen werden, dass es sich bei dem A* einfach nur um einen sehr genialen Algorithmus handelt, der sehr effizient und effektiv den kürzesten Pfad zwischen zwei Punkten

bestimmt, nicht mehr und nicht weniger. Auch mit Hilfe der *smooth* and *straight paths* können noch keine ästhetischen Pfade generiert werden, es sei denn eine Lokation umfasst sehr wenige, kleine, symmetrische Hindernisse (aber auch hierbei sollte davon ausgegangen werden, dass der generierte Pfad höchstwahrscheinlich nicht natürlich aussieht). Nutzt man den A* als Konzept, dann entstehen, aufgrund der Tatsache, dass der Pfad immer in Richtung Zielpunkt gedrängt wird, keine ästhetischen Pfade.

Da der A* aber als Algorithmus sehr entscheidend ist, wird nun auf die Implementierung seiner Komponenten eingegangen.

Umsetzung der A* Basis

Areas · Nachbarknoten · Grids · Datentypen · Knotenbank · Heuristik

Grundvoraussetzung

Es wurde festgestellt, dass zur effizienten und effektiven Implementierung bestimmte Voraussetzungen gegeben sein müssen:

- Lokationgrid,
- Heap Priority Queue,
- Hash Table
- Nodebank,
- Ermittlung der Knoten für die nächste Suchphase,
- Heuristik.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Des weiteren bedarf es ein paar Komponenten, die die A* Suche erst möglich bzw. schnell machen:

- Areas,
- Portale,
- Performanzkonzept: nur jeden 2. Punkt kontrollieren.

Areas

Prinzipiell werden die Laufbereiche der Lokations in sogenannte Areas eingeteilt.

Abbildung 28:
Laufareas (rote Linien),
es werden keine Pfade verwendet

Die Areas dienen dazu festzustellen, ob ein zu untersuchender Punkt begehbar ist. Ist dies nicht der Fall, dann wird der Punkt gar nicht erst beachtet.

Die Linien, die zwei benachbarte Areas gemein haben, nennen sich Portale. Bei der Vorstellung des neu entwickelten Pathfinding- Konzeptes wird man feststellen, dass diese Portale eine äußerst wichtige Rolle spielen.

Anhand eines Tools werden die Areas und Portale für die Engine in sogenannten res-Files (resourcen files) gespeichert, um mit ihnen in der Engine arbeiten zu können. Sie werden beispielsweise dazu verwendet festzustellen, wie groß ein Actor auf einem bestimmten Pixel skaliert oder welcher Schritt- Sound wann

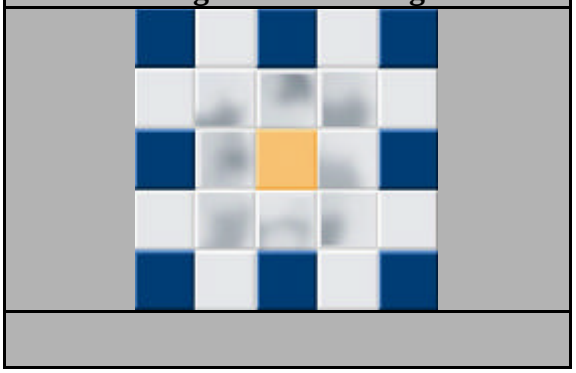
Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gespielt werden muss. Die Verwendung von Areas und Portalen ist in der 2D Spieleprogrammierung sehr beliebt.

Nachbarknotenermittlung

Die für die Diplomarbeit entwickelte Methode betrachtet nicht die direkten Nachbarknoten eines Knotens als Folgeknoten/ Successor, sondern jeden zweiten Pixel. Man braucht nicht zu erklären, dass sich der Suchaufwand dadurch drastisch verkleinert.

Abbildung 29:
Auszug einer Lokationgrid



Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Auszug des Pseudocode des A*:

```
....
while(Open ist nicht leer)
        {
                pop vielversprechendsten Knoten Node von Open
                if(Node ist Zielpunkt ptGoal)
                        Generiere Pfad und beende Suche
                else
                {
                        for(jeden neuen Nachbarknoten NewNode von Node)
                        <Erläuterung>
                        Hier müssen die Nachbarknoten zu Fortsetzung der A* Suche
                        ausgesucht werden
                        </Erläuterung>
                        ...
```

Die graumelierten Pixel um den gelben Suchpunkt, würden normalerweise als Nachbar- /Kindknoten des Suchknotens für die Suche verwendet werden. Zur Beschleunigung der Suche wurden die blauen Knoten verwendet. Um so vorgehen zu können, muss vor jeder Suche überprüft werden, ob der Zielpunkt vom Startpunkt erreichbar ist. Dies ist der Fall, wenn die x- Werte der beiden Punkte entweder beide gerade oder beide ungerade sind, denn nur so kann jeder zweite Punkt für die Suche verwendet werden; dies betrifft auch die y- Werte.

Pathfindingmap - Lokationgrid

An dieser Stelle sollte darauf hingewiesen werden, dass, neben der Ästhetik, im Sinne der Geschwindigkeit entwickelt wurde. Allgemein wurden somit folgende Prioritäten gesetzt: Ästhetik vor Geschwindigkeit vor Speicheraufwand. Prinzipiell bräuchte man nicht unbedingt eine Lokationgrid, weil man anhand eines Algorithmus feststellen könnte, ob sich ein Punkt in einer Area befindet. Betrachtet man den Algorithmus zur Feststellung des Enthaltenseins eines Punktes ein wenig genauer, dann wird einem bewusst, dass dieser Algorithmus nicht sonderlich schnell arbeitet.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Aus diesem Grund wurde beschlossen eine 800*600 Repräsentation der Lokation, also eine regular grid mit einem Pixel als Quadrat, aufzustellen, um mit O(1) festzustellen, ob ein Punkt begehbar ist. Es sei darauf hingewiesen, dass diese (eine) Grid von allen Charakteren benutzt wird, um zu vermeiden, dass zuviel Speicher verschwendet wird. Diese Grid wird dem Agenten wie beschrieben mit der Methode SetupWorldKnowledge(const GkCLocationGrid *m_WorldKnowledge, ..) übergeben.

Die Lokationgrid des Demoprogramms ist eine Ansammlung aller Bildschirmpixel. Da die Engine eine Auflösung von 800 * 600 Pixel erzwingt, besteht die Grid aus der Kombination von 800 x- und 600 y – Werten. Dies hat zur Folge, dass die Lokationgrid, wenn alle Kombinationen von x- und y – Werten vorhanden sein sollen, einen Speicherbereich für 800*600 = 480000 Pixel benötigt. Prinzipiell handelt es sich bei der Grid, wie gesagt, um eine regular grid, bei der ein Quadrat einem Pixel entspricht. Jeder dieser Pixel bedarf der Information, ob er begehbar ist. Aufgrund der Tatsache, dass im Beispielprogramm verschiedene Charaktere auf verschiedene Bodenbeläge unterschiedlich reagieren sollen (es gibt drei verschiedene Untergründe: Schnee, für die normalen Wege, Holz für Brücken, die über Wasser führen, und Eis), bedarf jeder Pixel der Information nichtbegehbar, Schnee, Holz oder Eis.

Würde man tatsächlich nicht mehr Informationen benötigen, dann hätte man einen Speicheraufwand von 4 bit / Pixel. Leider ist dies nicht so. Bei dem neu entwickelten System benötigt jeder Pixel die Nummer bzw. ID der Area, in der er sich befindet. Geht man davon aus, dass die Lokation des Demoprogramms, da sie sehr kompliziert ist, um die Stärken des Pathfindings besser herauszustellen, 25 – 30 Areas hat, normalerweise hat eine Lokation 4-5 Areas, dann bedarf ein Pixel einer ID von 0 – 30, was einem Speicheraufwand von 6 bit/ Pixel entspricht.

Es muss nicht darauf eingegangen werden, dass keine Strings für die Benennung der Areas verwendet werden, weil der Vergleich zu aufwendig wäre.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Da nun die Wertigkeiten der Pixelinformation bekannt sind, muss ein Datentyp für diese Information gewählt werden. Da versucht wird den Speicherbereich gering zu halten, sollte dieser Datentyp so klein wie möglich sein. Der kleinste Datentyp von einem Byte mit einem Wertebereich von –128 bis 127 (bzw. von 0 – 255) ist laut Microsoft Dokumentation __int8 (bzw. unsigned char). Benutzt man diesen Datentyp, dann bedarf die Lokationgrid eines Speicherbereiches von 800 * 600 * 1 Byte = 480000 Byte = 480 KByte. Bei einem Arbeitsspeicher von 32 MB, dieser entspricht dem einer PS2 oder eines PCs für Kinder, bedarf die Grid 480 * 100 KByte / 32000 KByte = 1,5 % des Speichers. (Es sei nur einmal erwähnt, dass die Grid bei einem Lokationwechsel neu generiert werden muss, weil jede Lokation andere Laufbereiche hat. Aus diesem Grund kennen die Agenten auch nur eine partielle Repräsentation ihrer Umwelt.) Da 1,5 % des Speicherbereiches relativ groß ist, könnte man ein Quadrat für den Bereich von 4, anstatt einem Pixel und die Begehbarkeit anhand der meist vorkommendsten Wertigkeit festlegen. Das Ziel dieses Pathfindingsystems ist jedoch die Berechnung der absolut kürzesten Pfade. Eine Einteilung in weniger Quadrate würde zu Abweichungen führen, die nicht gewünscht sind. Um trotzdem den Speicherbereich der Grid zu verkleinern, könnte man c++ bitfields verwenden. Ulrich Breymann rät von dieser Vorgehensweise ab, weil der „Aufwand des Zugriffes auf einzelne Bits beträchtlich sein kann, und nicht einmal sicher ist, ob wirklich Speicher gespart wird“[77]. Die Microsoft Dokumentation beschreibt keine dieser Gefahren, weist allerdings darauf hin, dass Zeiger oder Referenzen auf bitfields fehlerhaft sein können. Da keine direkten Zeiger und Referenzen auf die bitfields verwendet werden, besteht diese Gefahr für dieses System nicht. (In der Engine werden des Öfteren c/c++ bitfields verwendet.)

[77] Zitat: Ulrich Breymanns, c++ Eine Einführung, S.92

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Beispiel:

<source code>

```
typedef struct
{
        GkInt08 uiFloorCoveringType:6;
```

<Erläuterung>

6 bit bitfield.

</Erläuterung>

```
}MapNodeInformation;
```

</source code>

Mit den bitfields wird der Speicherbereich auf 0,75, d.h. ¾, reduziert, wenn ein 6 bit bitfield gewählt wird.

(800 * 600 * 1 Byte/ 4*3 = 360000 Byte = 360kByte => 1,333 %)

Zahlen zwischen 0 und 9 benötigen einen Speicherplatz von 4 bit, Zahlen zwischen 10 und 99 6 bit.

<source code>

```
class GkCLocationGrid
{
public: // Con- & Destruktor & operators
                GkCLocationGrid(const GkCLocationGrid &gkCLocationGrid);
                GkCLocationGrid(GkuInt32 ui32MaxXValues, GkuInt32 ui32MaxYValues);
                ~GkCLocationGrid();
                GkCLocationGrid &operator=(const GkCLocationGrid &gkCLocationGrid);

public:
        // Creates location Grid
        void Create(const char *lpszLocName, TwalkArea *tWalkArea, GkuInt08 ui08MaxAreas);

        // returns value if point is walkable
        GkuInt08 IsSolidFloorCoveringType(GkuInt32 ui32XValue, GkuInt32 ui32YValue) const;
        // returns floor covering type of special Point
        GkuInt08 GetFloorCoveringType(GkuInt32 ui32XValue, GkuInt32 ui32YValue) const;

        void SetFloorCoveringTypeAsLegal(GkuInt32 ui32XValue, GkuInt32 ui32YValue);
        void SetFloorCoveringTypeAsLegal(GkuInt32 ui32XValue, GkuInt32 ui32YValue, GkuInt08 ui08Type);

        // draws grid on the screen
        void DisplayGrid();
        // reset
```

```
        void Reset();

protected:
        // retuirn s fool specific covering type
        GkuInt08 GetFloorSpecificCoveringType(GkuInt08 ui08Floor);

private:
        GkuInt32 m_ui32MaxXValues;              // max X values of grid
        GkuInt32 m_ui32MaxYValues;              // max Y values of grid
        MapNodeInformation** m_GridMemory;// Grid memory
};
```
</source code>

Der Vorteil der regular grids ist, wenn sie wie gerade beschrieben verwendet werden, dass ein unheimlich schneller Zugriff von O(1) auf die Knoteninformation besteht. Die Abfrage kann mit den Methoden IsSolidFloorCoveringType(..) oder GetFloorCoveringType(..) getätigt werden. Beide Methoden geben Null zurück, wenn der Pixel nicht begehbar ist.

Erstellt wird die Grid anhand der Methode Create(const char *lpszLocName, TwalkArea *tWalkArea, GkuInt08 ui08MaxAreas), der der Lokationname, die Areas und die maximale Anzahl der Areas übergeben werden muss. Anhand dieser Methode wird die Grid einmalig anhand der Areas berechnet und dann in einem file abgespeichert (Filename = Lokationname + „.pmf“ [Beispiel für Demo: „..\PathfindingSample\res\Map\stadt.pmf“]). Beim nächsten Aufruf von Create(..) wird dann nur noch der file geladen, was die Erstellung der Map sehr stark beschleunigt. Bei der ersten Erstellung der Grid werden alle Pixel getestet, ob sie in einer Area liegen, falls ja, dann wird für den Pixel, wie erwähnt, die Areanummer gespeichert. Wenn ein Pixel nicht begehbar ist, dann erhält er die Wertigkeit 0.

Allgemein gibt es bei der Programmierung, einen Trade-Off zwischen Schnelligkeit und Speicherplatz. Die erläuterte Form der regular grids ist sehr schnell, verbraucht allerdings ziemlich viel Speicher. Die weniger speicherintensive Möglichkeit, bei der jeder Punkt permanent anhand der Areas während der Suche auf Begehbarkeit getestet wird, ist dagegen sehr langsam.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Rectangular grids sind die klassische Methode für die Suchmengendarstellung in 2D Welten. Die Erfahrung, die während dieser Diplomarbeit gemacht wurde, hat gezeigt, dass man eine solche Grid tatsächlich verwenden sollte, weil sie die Suche sehr stark beschleunigt. Die Grids bergen natürlich das Problem, dass nur ganzzahlige x- und y- Werte getestet werden können. Dies führt dazu, dass Gleitkommawerte gerundet werden müssen. Da die Grid sehr filigran aufgebaut ist, ist dies nicht so folgenschwer. Aus diesem Grund wird anhand von expliziten Typumwandlungsoperatoren gerundet.

```
float f = 1.005f;
int i = (int)f; // explizite Typumwandlung
```

Um zu einem genaueren Ergebnis zu kommen, müsste man „selber“ runden, d.h. den Tyumwandlungsoperator überschreiben, weil das System die Werte immer abrundet.

Hash Table für Open & Closed List - Priority Queue für Open List

Eine der umstrittensten Fragen im Bereich der Spieleprogrammierung ist, ob man die Standard Template Library (STL) verwenden sollte. Prinzipiell besteht die Meinung, dass man dies ohne Bedenken machen sollte. Fakt ist, dass die STL brillant implementiert ist und dass sie eine Vielzahl von Möglichkeiten bietet, die die Programmierung stark vereinfacht. Der Nachteil der STL ist, dass manche Plattformen, wie z.B. die Plattform der Dreamcastkonsole, die STL nicht unterstützen. Somit wäre es wahnsinnig die STL zu verwenden, wenn man nur im entferntesten vorhat, eine Engine zu entwickeln, mit der auch Konsolenspiele implementiert werden sollen. Aus diesem Grund wurde es vorgezogen, alle Datentypen selbst zu implementieren, um sie später ggf. noch verwenden zu können. Des weiteren hat man bei dieser Vorgehensweise eine viel bessere Kontrolle über die Datentypen.

Prinzipiell entstand das Problem die Operatoren zu überschreiben; aus diesem Grund wurden zum Teil Pseudooperatoren, wie beispielsweise IsEqual(..) anstatt operator=(..) , verwendet.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

<source code>

```
template<class T, class V>
class GkVCHashTable
{
public: // Con- & Destruktor
        GkVCHashTable(GkuInt32 ui32MaxSize = 100);
        ~GkVCHashTable();

protected: // virtual functions
        // Hash function
        // returns Number of hash adress
        virtual GkInt32 Hash(const V& vKey) = 0;

public:
        // function to enter data to the hash table
        // returns 1 if item could have been entered else 0
        GkuInt08 Enter(const T& tValue, const V& vKey, const GkuInt32 ui32HashValue);
        void ResetOpenOnly();

public:
        // alle Werte Reseten
        void Reset();
        // Find value and return HashKey
        // if item annot be found return -1
        GkInt32 Find(const V& vKey);
        // Get item
        const T& Get() const;

protected:
        struct HashCell
        {
                V* vKey;
                T tValue;
                HashCell* hNext;
        };

        GkuInt32 m_ui32MaxSize;          // max items of hash table
        HashCell** m_tDataMemory;                // hash table memory
        HashCell *m_pItem;
};


template<class T, class V>
class GkC2DPtHashTable: public GkVCHashTable<T, V>
{
public: // Con- & Destruktor
                GkC2DPtHashTable(GkuInt32 ui32MaxSize);
                ~GkC2DPtHashTable();
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
//public:
//        void EnterNodeData(const T& tNodeData, const V& vKey);

protected:
        // Hash function
        // returns Number of hash adress
        virtual GkInt32 Hash(const V& vKey);
};


template<class T, class V>
GkInt32 GkC2DPtHashTable<T, V>::Hash(const V& vKey)
{
        GkInt32 i32XValue = vKey.fXPos;
        GkInt32 i32YValue = vKey.fYPos;

        GkuInt32 iu32Hash = ((i32XValue  + i32YValue) % m_ui32MaxSize);
        return iu32Hash;
}
```

</source code>

Bei der Implementierung der Hash Table ist wahrscheinlich nur die Hashfunktion interessant. Die Hashfunktion bedarf sicherlich einer Optimierung, was dem gesamten Pathfindingsystem zugute kommen würde. Prinzipiell wurde eine klassische Hashumsetzung gewählt, weil der Augenmerk auf der ästhetischen Optimierung der Pfade lag. Die Notwendigkeit der Hash Table soll hier nicht noch mal erläutert werden.

<source code>

```
template<class T>
class GkCHeapPriorityQueue
{
public: // Kon-/ Destruktor
        GkCHeapPriorityQueue(GkuInt32 ui32MaxSize = 2000, GkuInt32 ui32Realoc = 500);
        ~GkCHeapPriorityQueue();

public:
        void Push(const T& tValue);
        GkuInt08 IsEmpty();
        T& Pop();
        void Update(const T& tValue);
        void Reset();
        T& Get(GkuInt32 ui32GetItem);
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
protected:
        void UpHeap(GkuInt32 ui32HeapEnd);
        void DownHeap(GkuInt32 ui32HeapEnd);
        void Swap(GkuInt32 ui32First, GkuInt32 ui32Second);
        // returns 0 if bigger
        // returns 1 if equal
        // returns 2 if smaller
        GkuInt08 SizeDueToParent(GkuInt32 ui32CheckItem);
        GkuInt08 CheckSiblings(GkuInt32 ui32Parent);
        GkuInt08 CheckHeap();


private:
        GkuInt32 m_ui32MaxSize;
        T *m_DataMemory;
        GkuInt32 m_ui32Iterator;
        GkuInt32 m_ui32Realoc;
};
```
</source code>

Die Priority Queue (PQ) hat fünf interessante Funktionen, wovon nur zwei die eigentliche Heap betreffen. Wofür die PQ notwendig ist wurde bereits in den vorangegangenen Kapiteln eingehend erläutert. Normalerweise wird eine PQ oft nach dem größten Wert sortiert. Aufgrund der Tatsache, dass der A* einer Bewertung der kürzesten Entfernung zum Zielpunkt bedarf, sortiert diese PQ nach dem kleinsten Wert.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Methoden:

- void Push(const T& tValue):

  Hiermit kann man einen Wert der PQ hinzufügen. Die PQ wird nach dem Hinzufügen hinsichtlich des kleinsten Wertes sortiert. Der Aufwand hierfür beträgt O(log n).

- T& Pop():

  Hiermit erhält man den kleinsten Wert der PQ, löscht diesen aus der PQ und sortiert die Queue neu. Der Aufwand hierfür ist O(log n).

- void Update(const T& tValue):

  Hiermit ändert man einen Wert und sortiert die Queue, wenn sich der Wert tatsächlich geändert hat, neu. Der Aufwand beträgt O(n + log n); hierbei wird überprüft, ob der Wert größer oder kleiner geworden ist, um die Neusortierung dementsprechend durchzuführen, um somit den Aufwand gering zu halten.

Die zwei interessanten Heap- Methoden sind:

- void UpHeap(GkuInt32 ui32HeapEnd):

  Sortiert von den Blättern des Baums in Richtung Wurzel (wobei der kleinste Wert die Wurzel darstellt).

- void DownHeap(GkuInt32 ui32HeapEnd):

  Sortiert in Richtung der Blätter.

Bei dieser Heap muss die Gesamtentfernung eines Kindknotens größer als die des Elternknotens und deren Schwesternknoten sein. Die Schwesternknoten werden, wie erwähnt, nicht sortiert, damit die Sortierung sehr schnell durchführbar ist.

<source code>

```
template<class T>
void GkCHeapPriorityQueue<T>::UpHeap(GkuInt32 ui32HeapEnd)
{
	// sort from back to force by the smallest
	if(m_ui32Iterator < 2)
		return;

	GkuInt32 ui32CurrentNode = ui32HeapEnd - 1;
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
	GkuInt32 ui32PriviousNode;
	while(ui32CurrentNode)
	{
		ui32PriviousNode = (ui32CurrentNode - 1) / 2;
		if(m_DataMemory[ui32PriviousNode]->IsSmallerThan(*m_DataMemory[ui32CurrentNode]))
			break;
		Swap(ui32PriviousNode, ui32CurrentNode);
		ui32CurrentNode = ui32PriviousNode;
	}
}

template<class T>
void GkCHeapPriorityQueue<T>::DownHeap(GkuInt32 ui32HeapEnd)
{
	// Sort from force to back by smallest
	GkuInt32 ui32Counter = ui32HeapEnd;

	if(m_ui32Iterator < 2)
		return;

	if(m_ui32Iterator == 2)
	{
		if(m_DataMemory[1]->IsSmallerOrEqualThan(*m_DataMemory[0]))
			Swap(1, 0);
	}
	else
	{
		GkuInt32 ui32SmallerNode = 0;
		GkuInt32 ui32NextNode = 0;
		GkuInt32 ui32PriviousNode = 0;

		while(ui32Counter <= m_ui32Iterator/2)
		{
			ui32PriviousNode = 2 * ui32Counter + 1;
			ui32NextNode = 2 * ui32Counter + 2;

			if(ui32NextNode >= m_ui32Iterator)
			{
				if(ui32NextNode == m_ui32Iterator)
				{
					if(!m_DataMemory[ui32Counter]-
					>IsSmallerOrEqualThan(*m_DataMemory[ui32PriviousNode]))
						Swap(ui32Counter, ui32PriviousNode);
				}

				break;
			}
```

```
                        if(m_DataMemory[ui32PriviousNode]-
                                >IsSmallerOrEqualThan(*m_DataMemory[ui32NextNode]))
                                ui32SmallerNode = ui32PriviousNode;
                        else
                                ui32SmallerNode = ui32NextNode;

                        if(m_DataMemory[ui32Counter]-
                                >IsSmallerOrEqualThan(*m_DataMemory[ui32SmallerNode]))
                                break;

                        Swap(ui32Counter, ui32SmallerNode);
                        ui32Counter = ui32SmallerNode;
                }
        }
}
```

</source code>

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Nodebank

Um nicht unnötig Speicher zu verschwenden, wurde eine Nodebank/ Knotenbank implementiert, bei der es sich um eine Vektorklasse handelt, die Speicherplatz für die Suche zur Verfügung stellt. Nur wenn ein Knoten/ Pixel tatsächlich für die Suche verwendet wird und dieser noch nicht in der Open oder Closed List repräsentiert ist, bekommt der Pixel Speicherplatz mit der Methode GkCVector<T>::GetFreeItem() zur Verfügung gestellt.

<source code>

```
template<class T>
class GkCVector
{
public: // Con- & Destruktor
        GkCVector(GkuInt32 ui32MaxSize, GkuInt32 ui32Realoc);
        ~GkCVector();
public:
        T& GetFreeItem();
        void Reset();
private:
        GkuInt32 m_ui32MaxSize;
        T *m_DataMemory;
        GkuInt32 m_ui32Iterator;
        GkuInt32 m_ui32Realoc;
};
```

</source code>

<source code>

```
class GkCPathfindingNode
{
public: // Con- & Destruktor & operator
                GkCPathfindingNode();
                ~GkCPathfindingNode();
                GkCPathfindingNode(const GkCPathfindingNode &gkCPathfindingNode);
                GkCPathfindingNode &operator=(const GkCPathfindingNode &gkCPathfindingNode);

public:
        void Reset();
        void ResetOpenOnly();

public:
        const GkCPathfindingNode *GetNodeParent() const;
        const GkPoint &GetPoint() const;
```

```
        const GkDouble GetCostFromStart() const;
        const GkDouble GetTotalCost() const;
        const GkInt08 IsOnOpen() const;
        const GkInt08 IsOnClosed() const;
        const GkDouble &GetGradient() const;
        const GkuInt08 GetIntersection() const;


        void SetNodeParent(GkCPathfindingNode* NodeParent);
        void SetPoint(const GkPoint& ptPoint);
        void SetCostFromStart(const GkDouble dCost);
        void SetTotalCost(const GkDouble dTotal);
        void SetOnOpen(const GkInt08 Int08OnOpen);
        void SetOnClosed(const GkInt08 Int08OnClosed);
        void SetGradient(const GkDouble dGradient);
        void SetIntersection(const GkuInt08 i08Intersection);


        const GkuInt08 IsEqual(const GkCPathfindingNode& gkCPathfindingNode) const;
        const GkuInt08 IsSmallerOrEqualThan(const GkCPathfindingNode& gkCPathfindingNode) const;
        const GkuInt08 IsSmallerThan(const GkCPathfindingNode& gkCPathfindingNode) const;
        const GkuInt08 IsGreaterOrEqualThan(const GkCPathfindingNode& gkCPathfindingNode) const;
        const GkuInt08 HasEqualTotalCosts(const GkCPathfindingNode& gkCPathfindingNode) const;

        GkInt08 GetLastArea() const;
        void SetLastArea(GkInt08 Int08LastArea);


private:
        void CopyMembers(const GkCPathfindingNode &gkCPathfindingNode);

private:
        GkCPathfindingNode* m_NodeParent;
        GkPoint m_Point;              // x, y- position of Node
        GkDouble m_dCost; // cost to get to this node
        GkDouble m_dTotal;// cost + heuristic
        GkInt08 m_Int08OnOpen;    // Node is on open list
        GkInt08 m_Int08OnClosed;  // Node is on close list
        GkInt08 m_Int08LastArea;  // kann nicht 0 sein
        GkDouble m_dGradient;
};
```

</source code>

Die Knoteninformationen sind sehr umfangreich. Während der Suche bedarf es des Wissens, wer der Vaterknoten des Knotens ist (GkCPathfindingNode::m_NodeParent), damit später sehr leicht der Laufpfad

zusammengesetzt werden kann. Des weiteren muss bekannt sein, um welchen Pixel/ Punkt es sich handelt (GkCPathfindingNode::m_Point). Eine weitere notwendige Information ist die Entfernung vom Startknoten bis zu diesem Knoten (GkCPathfindingNode::m_dCost), sowie die geschätzte Gesamtentfernung vom Startkoten über diesen Knoten zum Zielknoten (GkCPathfindingNode::m_dTotal). Weiterhin ist es notwendig zu wissen, ob sich der Knoten in der Open (GkCPathfindingNode::m_Int08OnOpen) oder Closed List (GkCPathfindingNode::m_Int08OnClosed) befindet. Speziell für das *forced portal A** Verfahren bedarf es der Information in welcher Area sich der Knoten befindet (GkCPathfindingNode::m_Int08LastArea) und welche Steigung der Knoten mit seinem Vater hat (GkCPathfindingNode::m_dGradient). Entscheidend ist, dass diese Informationen nur einmalig Speicherplatz reservieren dürfen, um den Speicheraufwand nicht zu groß werden zu lassen. Aus diesem Grund gibt es die Methode GkCPathfindingSystem::GetNode(..), die dafür sorgt, dass für jeden Pixel nur einmalig Speicherplatz reserviert wird. Dieser Speicherplatz wird bei jeder neuen Suche wieder vom Knoten entkoppelt.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

<source code>

```
__forceinline GkCPathfindingNode *GkCPathfindingSystem::GetNode(const GkFloat FloatStartXPosition, const GkFloat FloatStartYPosition)
{
	GkPoint ptPoint;
	ptPoint.fXPos = FloatStartXPosition;
	ptPoint.fYPos = FloatStartYPosition;

	// is start a legal point ?
	if(!m_WorldKnowledge->IsSolidFloorCoveringType(FloatStartXPosition, FloatStartYPosition))
		return NULL;

	GkCPathfindingNode *gkCNode = NULL;
	GkInt32 i32Hash = -1;
	if((i32Hash = m_MasterNodeHashTable->Find(ptPoint)) != -1)
	{
		gkCNode = &m_gkCNodeBankVector->GetFreeItem();
		gkCNode->SetPoint(ptPoint);
		gkCNode->SetOnOpen(0);
		gkCNode->SetOnClosed(0);
		if(!m_MasterNodeHashTable->Enter(gkCNode, gkCNode->GetPoint(), i32Hash))
			return NULL; // Error - hash table to small
	}
	else
		gkCNode = m_MasterNodeHashTable->Get();

	return gkCNode;
}
```

</source code>

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Umsetzung der Nachbarknotenermittlung

Die Ermittlung der Nachbar-/ Kindknoten wird mit der Methode GetSuccessors(..) durchgeführt. Um die Suche zu beschleunigen wird nur jeder 2. Knoten kontrolliert. Die ermittelten Punkte sind:

- Oberhalb: $n_1(x, y - 2)$
- oben rechts: $n_2(x + 2, y -2)$
- rechts: $n_3(x + 2, y)$
- unten rechts: $n_4(x + 2, y + 2)$
- unten: $n_5(x, y + 2)$
- unten links: $n_6(x - 2, y + 2)$
- links: $n_7(x - 2, y)$
- oben links: $n_8(x - 2, y -2)$

< source code >

```
__forceinline void GkCPathfindingSystem::GetSuccessors(const GkFloat FloatCurrentXPosition, const GkFloat
FloatCurrentYPosition, const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition)
{
	if(m_ui08PortalToPortalSearch)
	{
		...
	}
	else
	{
		GkuInt08 ui03Add = 2;
		if(!m_ui08EveryOtherNode)
			ui03Add = 1;

		GkPoint gkPoint;

		// above
		gkPoint.fXPos = FloatCurrentXPosition;
		gkPoint.fYPos = FloatCurrentYPosition - ui03Add;
		m_gkCSuccessors[0].SetPoint(gkPoint);

		// beyond
		gkPoint.fXPos = FloatCurrentXPosition;
		gkPoint.fYPos = FloatCurrentYPosition + ui03Add;
		m_gkCSuccessors[1].SetPoint(gkPoint);

		// left
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
                gkPoint.fXPos = FloatCurrentXPosition - ui03Add;
                gkPoint.fYPos = FloatCurrentYPosition;
                m_gkCSuccessors[2].SetPoint(gkPoint);

                // right
                gkPoint.fXPos = FloatCurrentXPosition + ui03Add;
                gkPoint.fYPos = FloatCurrentYPosition;
                m_gkCSuccessors[3].SetPoint(gkPoint);

                // beyond left
                gkPoint.fXPos = FloatCurrentXPosition - ui03Add;
                gkPoint.fYPos = FloatCurrentYPosition + ui03Add;
                m_gkCSuccessors[4].SetPoint(gkPoint);

                // beyond right
                gkPoint.fXPos = FloatCurrentXPosition + ui03Add;
                gkPoint.fYPos = FloatCurrentYPosition + ui03Add;
                m_gkCSuccessors[5].SetPoint(gkPoint);

                // above left
                gkPoint.fXPos = FloatCurrentXPosition - ui03Add;
                gkPoint.fYPos = FloatCurrentYPosition - ui03Add;
                m_gkCSuccessors[6].SetPoint(gkPoint);

                // above right
                gkPoint.fXPos = FloatCurrentXPosition + ui03Add;
                gkPoint.fYPos = FloatCurrentYPosition - ui03Add;
                m_gkCSuccessors[7].SetPoint(gkPoint);
        }
}
```

< /source code >

Heuristik

Das Entscheidendste beim A*- Algorithmus ist die heuristische Abschätzung der Entfernung. Nur wenn die Heuristik richtig gewählt wird, bewegt sich die Suche auf den Zielpunkt zu und ist effektiv und effizient. Man muss darauf achten, dass die Heuristik weniger als die eigentliche Entfernung ist, um ein gutes Suchergebnis zu erhalten. Aus diesem Grund verwendet man den Satz des Pythagoras zur Ermittlung der Entfernung zwischen zwei Punkten: $a^2 + b^2 = c^2$.

```
GkFloat fXd = gkcFirstNode.GetPoint().fXPos - fSecondXPosition;
GkFloat fYd = gkcFirstNode.GetPoint().fYPos - fSecondYPosition;
GkDouble dDistance = (fXd * fXd) + (fYd * fYd);
```

Es sollte vermieden werden die Wurzel der Distanz (c = dDistance) zu ziehen, weil dies die Performanz zu stark beeinflussen würde.

Vermeiden:
c = ?($a^2 + b^2$) bzw. GkDouble dDistance = ? ((fXd * fXd) + (fYd * fYd));

Da alle Berechnungen den Satz des Pythagoras verwenden, ist das „Wurzel – Ziehen“ auch nicht notwendig.

Da der Pseudocode für den A* schon erläutert wurde, soll der eigentliche, für die Diplomarbeit implementierte A* erst dann erklärt werden, wenn das Pathfindingkonzept vorgestellt wurde, weil er ein paar Besonderheiten aufweist. Prinzipiell wurde bis hierhin dargestellt, was tatsächlich alles für ein Pathfindingsystem, welches sich auf den A* stützt, benötigt wird und wie dies im Detail umzusetzen ist. Prinzipiell benötigt man Areas und Portale, um zu wissen, wo gegangen werden darf. Des weiteren benötigt man eine Lokationgrid, um schnell festzustellen welche Punkte begehbar sind. Außerdem bedarf man ein paar schneller Datentypen, einer guten Heuristik und einer Knotendatenbank.

Autor: René Greulich Matrikelnr.: 110 18781
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke Zweitprüfer: Prof. Dr. E. Ehses

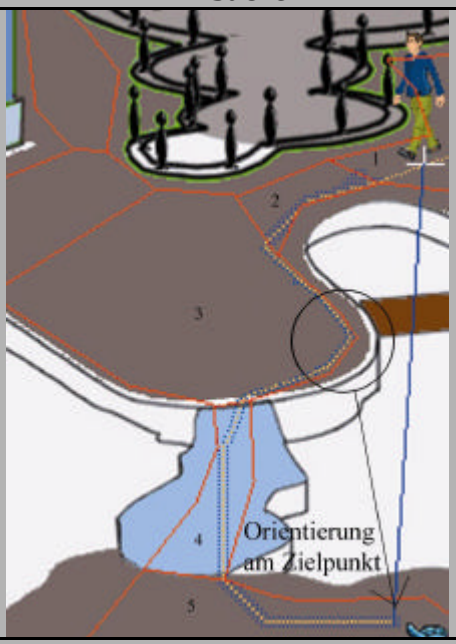
Forced Portal A* Search - Das neue Pathfindingkonzept

Die A Erkenntnis*

A* allein ist noch keine Pfadfindungslösung

Wie schon erwähnt wurde angenommen, dass der A* ausreicht, um ein gutes ästhetisches, schnelles Pathfinding zu implementieren. Fakt ist, dass das Pathfinding mittels des A* wirklich schnell ist, dass der ermittelte Pfad meistens allerdings nicht sonderlich ästhetisch aussieht.

Abbildung 30:
Unästhetisches Ergebnis der normalen A* Suche



Der Nachteil des A*, der gleichzeitig sein Vorteil ist, ist, dass die Heuristik, wie man sieht, wie ein Magnet funktioniert. Egal welchen Punkt des Pfades man wählt, sein Nachfolger orientiert sich immer in Richtung des Zielpunktes. Wie man sieht verläuft der Pfad entlang der Aussenkante (in Richtung Zielpunkt) der Area. Ein Mensch würde nicht diesem Pfad folgen, sondern sich an dem Portal (in diesem Fall

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

an dem Portal zwischen Area 3 und Area 4), welches er als nächstes durchlaufen möchte, orientieren, d.h. er würde einen direkteren, kürzeren Pfad durch die Area 3 wählen.

Versucht man nun die Heuristik zu verändern, indem man sie durch Faktoren wie 0.8f und 1.2f verringert oder erhöht, dann ergibt sich auch kein besserer Pfad. Der Pfad orientiert sich zu stark am Zielpunkt und sieht sehr künstlich aus.

Beispiel:

1. GkDouble dDistance = ((fXd * fXd) + (fYd * fYd)) * 0.8f;
2. GkDouble dDistance = ((fXd * fXd) + (fYd * fYd)) * 1.2f;

Die erste Idee, die danach entwickelt wurde, war, den Pfad zu begradigen, um einen natürlicheren Weg zu generieren. Aus diesem Grund entstand die Methode AddCostToNonStraightPath(..), die sich allgemein als sehr nützlich erwiesen hat. Mit Hilfe dieser Methode kann ermittelt werden, ob der momentan untersuchte Knoten die gleiche Steigung mit seinem Vaterknoten besitzt, wie der Vaterknoten zu seinem Vaterknoten. Kurz: hier wird geprüft, ob die drei Knoten auf einer Gerade liegen. Ist dies der Fall, dann wird 0 zurückgegeben und es wird kein *Penalty* der Heuristik hinzugefügt, ansonsten wird 1 zurückgegeben und ein *Penalty* aufgerechnet. Schon hier sei erwähnt, dass bewusst mit den Steigungen gearbeitet wird, weil diese später noch benötigt werden. (Prinzipiell könnte der Knoten einfach in die Geradengleichung des Elternknotens eingesetzt werden, um zu testen, ob die Knoten die gleiche Steigung haben. Sollte dies allerdings nicht der Fall sein, dann müsste die neue Steigung zusätzlich berechnet werden, was performanztechnisch nicht vorteilhaft wäre. Somit wird beim *forced portal A** sofort die Steigung berechnet und diese mit der Steigung des Vaterknotens verglichen.)

<source code >

```
__forceinline GkDouble GkCPathfindingSystem::AddCostToNonStraightPath(GkCPathfindingNode &gkcCurrentNode, const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition)
{
	if(gkcCurrentNode.GetNodeParent())
	{
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
const GkCPathfindingNode *gkcParentOfParentNode;
if(gkcCurrentNode.GetNodeParent()->GetNodeParent())
        gkcParentOfParentNode = gkcCurrentNode.GetNodeParent()->GetNodeParent();
else
        gkcParentOfParentNode = GetNode(FloatGoalXPosition, FloatGoalYPosition);

if(gkcParentOfParentNode)
{
        if(!gkcCurrentNode.GetNodeParent()->GetNodeParent())
        {
                GkDouble dDeltaParentsX = gkcParentOfParentNode->GetPoint().fXPos –
                        gkcCurrentNode.GetNodeParent()->GetPoint().fXPos;
                GkDouble dDeltaParentsY = gkcParentOfParentNode->GetPoint().fYPos –
                        gkcCurrentNode.GetNodeParent()->GetPoint().fYPos;

                GkDouble dDeltaNodeToGoalX = gkcCurrentNode.GetNodeParent()-
                        >GetPoint().fXPos - gkcCurrentNode.GetPoint().fXPos;
                GkDouble dDeltaNodeToGoalY = gkcCurrentNode.GetNodeParent()-
                        >GetPoint().fYPos - gkcCurrentNode.GetPoint().fYPos;

                if(!dDeltaParentsX || !dDeltaNodeToGoalX)
                {
                        dDeltaParentsX += 0.0000001;
                        dDeltaParentsY += 0.0000001;
                        dDeltaNodeToGoalX += 0.0000001;
                        dDeltaNodeToGoalY += 0.0000001;
                }

                gkcCurrentNode.SetGradient((GkDouble)((GkDouble)dDeltaNodeToGoalY/
                        (GkDouble)dDeltaNodeToGoalX));
                if(((GkDouble)(dDeltaParentsY/ dDeltaParentsX)) ==
                        ((GkDouble)(dDeltaNodeToGoalY/ dDeltaNodeToGoalX)))
                        return 0;
        }
        else
        {
                GkDouble dDeltaNodeToGoalX = gkcCurrentNode.GetNodeParent()-
                        >GetPoint().fXPos - gkcCurrentNode.GetPoint().fXPos;
                GkDouble dDeltaNodeToGoalY = gkcCurrentNode.GetNodeParent()-
                        >GetPoint().fYPos - gkcCurrentNode.GetPoint().fYPos;

                if(!dDeltaNodeToGoalX)
                {
                        dDeltaNodeToGoalX += 0.0000001;
                        dDeltaNodeToGoalY += 0.0000001;
                }

                GkDouble m = (GkDouble)((GkDouble)dDeltaNodeToGoalY/
```

```
                                        (GkDouble)dDeltaNodeToGoalX);
                                    gkcCurrentNode.SetGradient(m);

                                    if((gkcCurrentNode.GetGradient()) == (gkcCurrentNode.GetNodeParent()-
                                        >GetGradient()))
                                        return 0;
                                }
                            }
                        }

                        return 1;
}
```

< /source code >

Der Sinn der Begradigung ist, wie schon erwähnt, die Ermittlung des schönsten möglichen Weges, weil meist mehrere Wege von A nach B mit der gleichen Länge vorhanden sind. Da eine regular grid verwendet wird, addiert man 10^{-7} zur Heuristik, wenn mittels AddCostToNonStraightPath(..) festgestellt wurde, dass der Suchknoten, sein Vaterknoten und dessen Vaterknoten nicht auf einer Geraden liegen:

```
GkDouble dDistance = (fXd * fXd) + (fYd * fYd) + 0.0000001;
```

Es war erkennbar, dass ein minimal besserer Weg entstand, dass das Grundproblem hiermit aber nicht gelöst wurde. Die Addition verbesserte vielleicht 10 Pixel. Prinzipiell sah der Pfad aber immer noch gleich aus. Aus diesem Grund wurde danach die Hälfte der Distanz auf die Distanz addiert:

```
GkDouble dDistance = ((fXd * fXd) + (fYd * fYd)) * 1.5f;
```

Das Ergebnis unterschied sich nicht von der Addition von 10^{-7}. Somit erlangte man immerhin die Bestätigung, dass die Addition von 10^{-7} in einer regular grid tatsächlich ausreicht, um eine Begradigung zu erreichen. Das Grundproblem bestand aber immer noch.

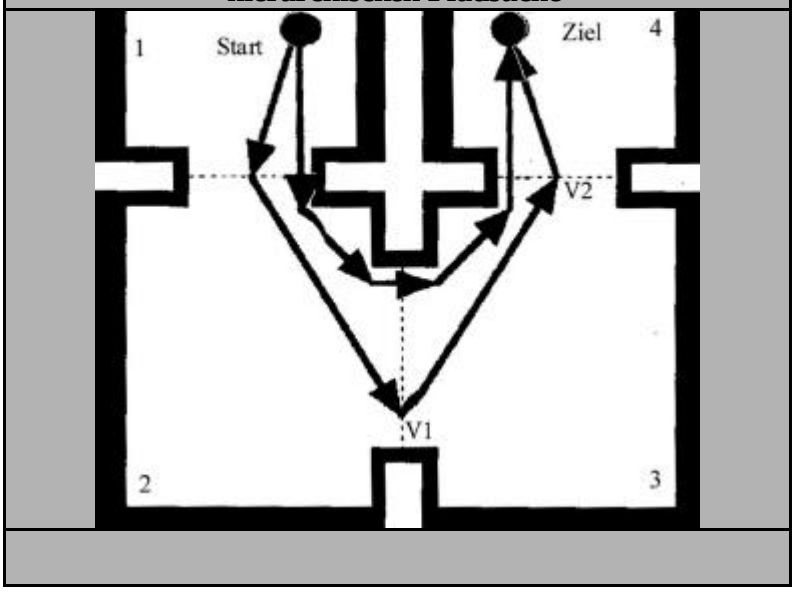
Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Da das *smoothing* mittels der *Catmull Methode* nur vorhandene Punkte verarbeitet, um einen kurvenförmigeren Pfad zu erhalten, musste dies, um einen schöneren Pfad zu generieren, erst einmal nicht versucht werden.

Hierarchisches Pathfinding

Beim *hierarchischen Pathfinding* wird erst einmal, anhand der Mittelpunkte von Portalen, der ungefähre Pfad vom Start zum Ziel ermittelt, um dann die Pfade zwischen den Portalen zu generieren. Damit die entstehenden Pfade auch realistisch aussehen und nicht nur durch die Mittelpunkte der Portale gehen, ist der Zielpunkt, bei der Ermittlung der Pfade zwischen den Portalen, der Mittelpunkt des übernächsten Portals (V1). Wenn das nächste Portal durchlaufen wird (Portal zwischen Area 1 und 2 (siehe untere Abbildung)), dann beginnt eine neue Suche, wieder in Richtung des übernächsten Portals (V2), usw. Sollte kein übernächstes Portal mehr vorhanden sein, dann ist der eigentliche Zielpunkt das Ziel.

Abbildung 31:
Genereller und idealer Weg bei der hierarchischen Pfadsuche



Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Um diese Methode umzusetzen, war es erst einmal notwendig eine Portalklasse zu erstellen:

<source code >

```
class GkCPortal
{
public:
        GkCPortal();
        GkCPortal(const GkCPortal &gkCPortal);
        ~GkCPortal();
        GkCPortal &operator=(const GkCPortal &gkCPortal);

public:
        void SetUp(GkFloat fXPosPoint1, GkFloat fYPosPoint1, GkFloat fXPosPoint2, GkFloat fYPosPoint2, GkuInt08
ui08AreaNr);
        void ChoosePointMiddle(const GkFloat fGoalXPosition, const GkFloat fGoalYPosition);

        const GkPoint &GetPointOne() const;
        const GkPoint &GetPointTwo() const;
        GkDouble GetGradient() const ;
        GkDouble GetYAchsenAbschnitt() const;

        const GkFloat GetMaxX() const;
        const GkFloat GetMinX() const;

        const GkFloat GetMaxY() const;
        const GkFloat GetMinY() const;
        const GkPoint &GetPointMiddle() const;

        const GkuInt08 IsArea(GkuInt08 ui08AreaNr) const;
        const GkInt08 GetOtherArea(GkuInt08 ui08AreaNr) const;

        const GkInt08 GetIterator() const;

private:
        GkPoint m_PointMiddle;
        GkPoint m_PointMiddleOne;
        GkPoint m_PointMiddleTwo;

        GkPoint m_PointOne;
        GkPoint m_PointTwo;
        GkDouble m_m; // Steigung
        GkDouble m_b; // y Achsen - Abschnitt
        GkFloat m_fMinX;
        GkFloat m_fMaxX;
        GkFloat m_fMinY;
```

```
        GkFloat m_fMaxY;
        GkuInt08 *m_ui08Area;
        GkuInt08 m_ui08AreaIterator;
        GkuInt08 m_ui08UsedPortalPoint; // 0 = PointOne, 1 = PointMiddle ,2 = PointTwo
};
```

< /source code >

Des weiteren benötigte man eine Portalkontrollklasse, um besser mit den Portalen und Areas arbeiten zu können.

< source code >

```
typedef struct
{
        GkCPortal **gkCPortal;
        GkuInt08 uiIterator;
}GkSArea;

class GkCPortalManager
{
public: // Con- & Destruktor & operators
                GkCPortalManager(GkuInt32 ui32MaxPortals, GkuInt32 ui32MaxAreas, GkuInt32
ui32MaxPortalsInArea);
                ~GkCPortalManager();

public:
        // Enter Portal
        void EnterPortal(GkFloat fXPosPoint1, GkFloat fYPosPoint1, GkFloat fXPosPoint2, GkFloat fYPosPoint2, GkuInt08
ui08AreaNr);
        // Get Portal
        const GkCPortal* GetPortal(const GkuInt32 uiPortalNum) const;
        GkCPortal* GetPortal(GkInt08 i08Area, GkFloat FloatCurrentXPosition, GkFloat FloatCurrentYPosition) const;
        // create areas
        void CreateAreas();
        // Get Area
        const GkSArea* GetArea(const GkuInt32 uiAreaNum) const;
        // Get other area
        const GkuInt08 GetOtherArea(GkInt08 i08Area, GkFloat FloatCurrentXPosition, GkFloat FloatCurrentYPosition)
const;
        // CheckPortals if portals have two areas
        void CheckPortals();

private:
        GkuInt32 m_ui32MaxPortals;
        GkuInt32 m_uiIterator;
        GkCPortal *m_gkCPortal;
```

```
        GkSArea *m_gkSArea;
        GkuInt32 m_ui32MaxPortalsInArea;
        GkuInt32 m_ui32MaxAreas;
};
```

< /source code >

Mit Hilfe dieser Klassen war es möglich eine reine Portal zu Portal Suche[78] durchzuführen. In diesem Fall war es notwendig die GetSuccessors(..)- Methode anzupassen, weil jetzt nicht mehr die benachbarten Knoten eines Knotens zur weiteren Suche verwendet werden, sondern die Mittelpunkte der Portale, die zur selben Area gehören, wie der Mittelpunkt (Suchknoten) des momentan betrachteten Portals. An der Heuristik musste nichts verändert werden. Mit Hilfe dieser Portalmittelpunkte kann nach der reinen Portal zu Portal Suche, der Gesamtpfad generiert werden.

Prinzipiell errechnet diese Methode einen schöneren Weg. Bei genauerer Betrachtung zeigt sich jedoch, dass das Grundproblem, der Magnetismus der Heuristik, mit dieser Methode keineswegs gelöst wird. Die Pfade wirken immer noch sehr unnatürlich, auch wenn man die *straight path* Methode anwendet.

Die erfolglosen Bemühungen haben gezeigt, dass sowohl die reine A* Suche, als auch die hierarchische A* Suche, für ästhetische Pfade nicht die richtige Lösung und dass der A* wirklich nur ein Algorithmus ist. Somit bestand das neue Ziel in der Entwicklung einer neuartigen Methodik oder eines Pathfindingkonzeptes, basierend auf dem A*, um die extremen Anforderungen an die Ästhetik zu befriedigen. Natürlich ist es prinzipiell möglich nur mit dem A* oder dem *hierarchischen Pathfinding* vorzugehen, aber nicht in Hinsicht auf die *phänomenale Gestalt*.

[78] *reine Portal zu Portal Suche*: pure portal to portal search; Ermittlung des generellen Weges der hierarchischen Suche.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Forced Portal A Search*

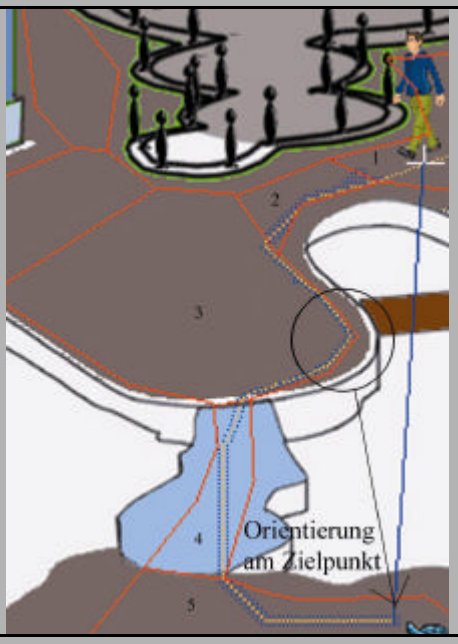
Die neue Methode

Tatsache ist, dass die Heuristik maßgeblich für eine schnelle Suche ist. Ungünstigerweise entwickelt sie die Eigenschaft, den Weg unnatürlich in Richtung des Zielpunktes zu drängen. Aus dieser Feststellung entstand die Erkenntnis, dass eine zweite Kraft wirken muss, die die Kraft der Heuristik aufhebt, besser gesagt in eine andere Richtung lenkt.

Das *hierarchische Pathfinding* wird dazu verwendet die A* Suche zu beschleunigen, indem zuerst der allgemeine Weg über die Mittelpunkte von Portalen zwischen den Areas berechnet wird und anhand dieser Punkte während des Laufens die Pfade zwischen den Portalpunkten berechnet werden. Das für die Diplomarbeit entwickelte Pathfindingkonzept adaptiert diese Methode in gewissem Sinn, denn auch bei der *forced portal A** Suche muss erst einmal der allgemeine Weg bekannt sein.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Abbildung 32:
Unästhetisches Ergebnis der normalen A* Suche

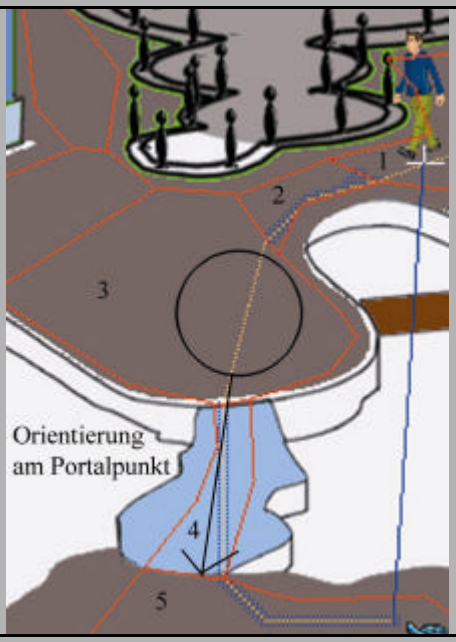


Das Problem der normalen A* Suche ist, dass eine Orientierung am Zielpunkt stattfindet. Wenn der Charakter von der Area 1 über 2, 3 und 4 nach 5 laufen soll, dann wird mit Hilfe des A* in der Area 3 kein direkter Pfad von 2 nach 4 erstellt. Das Problem ist die heuristische Abschätzung zum Zielpunkt. Aufgrund der direkten (kurzen) Entfernung über das Hindernis zum Zielpunkt werden immer alle Punkte am Rand der Area 3 für den kürzesten Weg verwendet und niemals ein direkter Weg zu Area 4 ermittelt.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Eine bessere Lösung würde entstehen, wenn in diesem Fall die *hierarchische Suche* verwendet werden würde, weil dann eine Orientierung am Mittelpunkt des Portals zwischen der Area 4 und 5 stattfinden würde, was zu einem natürlichen Pfad führen würde.

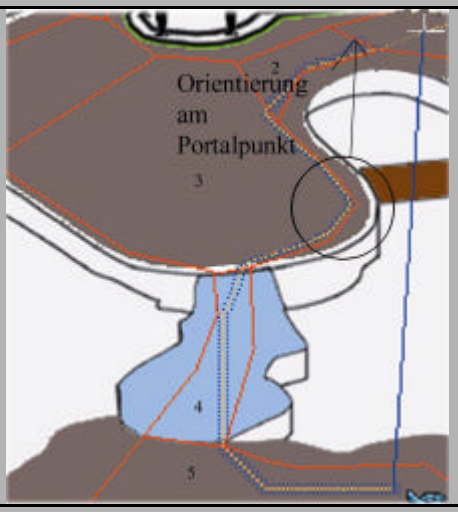
Abbildung 33:
besseres Ergebnis mit hierarchischer A* Suche



Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Das die *hierarchische Suche* zu einem besseren Ergebnis führt liegt allerdings nur an den äußeren Umständen. Die Portale zwischen 3 und 4 und 4 und 5 sind von ihrer Anordnung so ähnlich, dass dies zu einem guten Ergebnis führen muss. Das Problem ist, dass die Portale beinahe parallel verlaufen und fast gleichgroß sind und es somit gleichgültig ist, ob man das erste oder das zweite Portal als Ziel verwendet. Prinzipiell bräuchte man das zweite Portal gar nicht in die Suche einbeziehen und würde (beinahe) das gleiche Ergebnis erhalten. Versucht man den umgekehrten Weg von 5 über 4,3 und 2 nach 1 zu generieren, dann steht man wieder vor der gleichen Problematik wie zuvor, weil der Weg anhand der Heuristik über das Hindernis wesentlich kürzer ist.

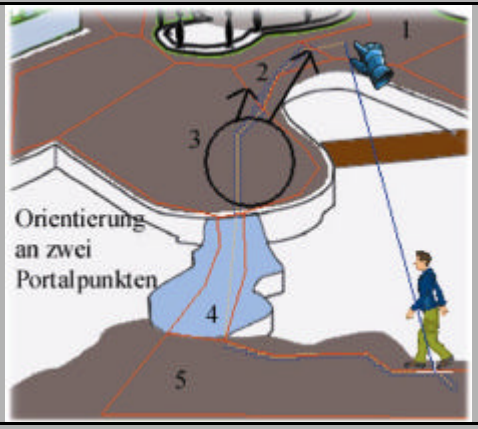
Abbildung 34:
schlechtes Ergebnis mit hierarchischer A* Suche



Um diese Problematik abzufangen muss man die *forced portal A** Suche durchführen. Hierbei findet nicht nur eine Orientierung anhand des vom aktuellen Suchpunkt entfernten zweiten Portals (Portal zwischen Area 1 und 2) statt, sondern man orientiert sich auch am nächsten Portal (Portal zwischen 2 und 3). Wie bei der nächsten Abbildung zu erkennen ist, wird mit dieser Methode der perfekte ästhetische Pfad berechnet.

Autor: René Greulich Matrikelnr.: 110 18781
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke Zweitprüfer: Prof. Dr. E. Ehses

Abbildung 35:
Lösung: forced portal
A* Suche



Bestandteile der forced portal A* search

Die forced portal A* Suche unterteilt sich in vier grobe Verfahren:

- pure portal to portal search (PPTP)
- portal to portal search (PTP)
- last portal search (LPS)
- goal area search (GAS)

Die eigentliche A* Suche ist bei allen vier Verfahren gleich. Sie entspricht in etwa dem vorgestellten Pseudocode, wird aber nochmals detailliert dargestellt. Wird von nun an von der *normalen A* Suche* gesprochen, dann bedeutet dies, dass die Heuristik das normale Ergebnis des Satz des Pythagoras ist.

```
GkDouble dDistance = (fXd * fXd) + (fYd * fYd)
```

Ansonsten wird die Heuristik unter bestimmten Umständen mit einem Faktor manipuliert, um die Suche in eine bestimmte Richtung zu drängen.

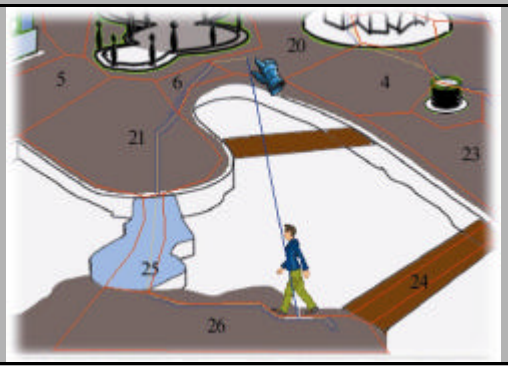
Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Pure Portal To Portal Search (PPTP)

Beim PPTP wird tatsächlich nur die normale A* Suche mit der unmanipulierten Heuristik verwendet. Wie schon erwähnt wird hierbei anhand der Mittelpunkte aller Portale der kürzeste Weg zwischen dem Start- und Zielpunkt ermittelt. Ziel der PPTP ist die Ermittlung dieser Mittelpunkte und der Areas, durch die der kürzeste Weg führen soll. Es sollte darauf hingewiesen werden, dass wirklich nur die Portalpunkte verwendet werden, keine normalen Areapunkte. Mit der Methode GetSuccessors(..) werden dann auch nur Kindknoten erstellt, die Mittelpunkte der Portale der gleichen Area sind. Da normalerweise acht Kindknoten erstellt werden, d.h. es wird ein Speicherplatz für acht Punkte angelegt, die Areas meist aber weitaus weniger Portale besitzen, werden bei allen nichtverwendeten Speicherplätzen, die x-Werte auf –1 gesetzt, um später eine Abfragemöglichkeit zu haben, ob die Knoten für die A* Suche verwendet werden sollen. Da man sich in einem Bereich von 800*600 bewegt, kann unter normalen Umständen der x- Wert niemals –1 sein.

Die PPTP Suche sieht für dieses Beispiel folgendermaßen aus (Auszug aus dem Debug- File):

Abbildung 36:
Beispiel forced portal
A* Suche



Der eigentliche Start- und Zielpunkt:

Start: X:216.00000000 Y:569.00000000

Goal: X:233.00000000 Y:347.00000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Diese Punkte müssen angepasst werden, damit später bei der Suche jeder 2. Punkt verwendet werden kann. Somit ändert sich der Zielpunkt:
Start: X:216.00000000 Y:569.00000000
Goal: X:232.00000000 Y:347.00000000

Als nächstes wird der Startpunkt der Open List hinzugefügt:
plane portal search add start node

Die A* Suche beginnt:
starting plane portal search

Der erste beste Knoten aus der Open List ist natürlich der Startknoten:
bestnode: X:216.00000000 Y:569.00000000 Total:49540.00000000

Dieser Knoten hat zwei mögliche Nachbarknoten, den Mittelpunkt des Portals zwischen den Areas 25 und 26 und den Mittelpunkt des Portals zwischen den Areas 24 und 26:
gkCActualNode: X:122.00000000 Y:545.50000000 Total:60890.50000000
gkCActualNode: X:319.00000000 Y:570.50000000 Total:68132.50000000

Es wird sich für den Mittelpunkt des Portals zwischen den Areas 25 und 26 entschieden, weil die geschätzte Gesamtentfernung (60890.5) kleiner als die Gesamtentfernung (68132.5) des anderen Portalpunktes ist.

Bester Knoten:
bestnode: X:122.00000000 Y:545.50000000 Total:60890.50000000
Dieser beste Knoten hat einen Nachbarknoten, den Mittelpunkt des Portals der Areas 21 und 25. Der Portalknoten zwischen 24 und 26, der auch ein Nachbarknoten wäre, wird ignoriert, weil nur Portalknoten der Area 25 betrachtet werden:
gkCActualNode: X:144.50000000 Y:464.50000000 Total:37918.00000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Bester Knoten:

bestnode: X:144.50000000 Y:464.50000000 Total:37918.00000000

Der beste Knoten hat zwei mögliche Nachbarknoten, den Mittelpunkt des Portals zwischen den Areas 6 und 21 und den Portalknoten zwischen den Areas 5 und 21

gkCActualNode: X:156.00000000 Y:382.00000000 Total:30395.00000000

gkCActualNode: X:69.00000000 Y:389.50000000 Total:56156.00000000

Beste Knoten:

bestnode: X:156.00000000 Y:382.00000000 Total:30395.00000000

Der beste Knoten hat einen möglichen Nachbarknoten, den Portalknoten zwischen den Areas 20 und 6:

gkCActualNode: X:200.00000000 Y:356.50000000 Total:27094.50000000

Dieser beste Knoten ist in der Area des Zielknotens, deshalb ist die PPTP hier zu Ende

bestnode: X:200.00000000 Y:356.50000000 Total:27094.50000000

CreatePortalPath start

nArea:6 X:200.00000000 Y:356.50000000

nArea:21 X:156.00000000 Y:382.00000000

nArea:25 X:144.50000000 Y:464.50000000

nArea:26 X:122.00000000 Y:545.50000000

CreatePortalPath done

Der PPTP- Pfad besteht aus den Portalpunkten A_{26}(122, 545.5), A_{25}(144.5, 464.5), A_{21}(156, 382), A_6(200, 356.5) und führt durch die Portale 26, 25, 21, 6.

Genau nach diesem Prinzip funktioniert die A* Suche. Nur die verwendeten Punkte variieren. Die restliche Debug- Ausgabe für diese Suche kann man sich im Anhang ohne Kommentare ansehen. Sie wurde hier nicht weiter aufgeführt, weil sie sehr umfangreich ist.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Portal To Portal Search (PTP)

Nachdem nun die Portalpunkte und die Areas bekannt sind, kann die eigentliche *forced portal A** Suche beginnen. Prinzipiell wird jetzt folgendes durchgeführt:

- Suche einen Weg (mittels dem A*) vom Start (216, 569) zum übernächsten Portalpunkt A_{25} (144.5, 464.5); achte bei dieser Suche darauf, dass ein Pfad gewählt wird, der in Richtung des nächsten Portals A_{26}(122, 545.5) liegt. Sobald die Area 25 erreicht ist, merke dir diesen ersten Punkt der Area 25, weil er der neue Startpunkt ist und stoppe diese Suche.
- Suche einen Weg (mittels dem A*) vom neuen Startpunkt zum übernächsten Portalpunkt A_{21}(156, 382); achte bei dieser Suche darauf, dass ein Pfad gewählt wird, der in Richtung des nächsten Portals A_{25} (144.5, 464.5) liegt. Sobald die Area 21 erreicht ist, merke dir diesen ersten Punkt der Area 21, weil er der neue Startpunkt ist und stoppe diese Suche.
- Suche einen Weg (mittels dem A*) vom neuen Startpunkt zum übernächsten Portalpunkt A_6(200, 356.5); achte bei dieser Suche darauf, dass ein Pfad gewählt wird, der in Richtung des nächsten Portals A_{21}(156, 382) liegt. Sobald die Area 6 erreicht ist, merke dir diesen ersten Punkt der Area 6, weil er der neue Startpunkt ist und stoppe diese Suche.

Die PTP wäre hiermit beendet. Der Vorteil, weshalb man das übernächste Portal und nicht das nächste als Zielpunkt betrachtet, ist, dass ein natürlicher Weg ermittelt wird, weil vermieden wird, dass alle Pfade durch die Mittelpunkte der Portale führen.

Die bedeutende Frage ist, wie dafür gesorgt wird, dass, im Gegensatz zum normalen A* und der hierarchischen A* Suche, tatsächlich ein Weg generiert wird, der durch die nächsten Portale führt. Es muss eine Gegenkraft zur heuristischen Abschätzung der Entfernung zum Zielpunkt, in diesem Fall zum übernächsten Portalpunkt, geschaffen werden. Beim A* kann, wie an der *straight path* Methode erläutert wurde, eine Einflussnahme auf die Wegführung genommen werden, indem

man, im Falle einer falschen Wegführung, mittels eines Faktors die Heuristik dermaßen überschätzt, dass die Entfernung nicht mehr lukrativ erscheint. Hier beginnt die Voodoo- Vorgehensweise von der Steve Rabin geredet hat. Prinzipiell gibt es keinen logischen Faktor, was daran liegt, dass die Areas keinem Standard folgen. Man muss durch Ausprobieren herausfinden, welcher Penaltyfaktor der Richtige ist. Prinzipiell wäre es durchaus möglich, sich mehrere Wochen nur mit der Beeinflussung der Heuristik zu beschäftigen, weil dieser Faktor nicht nur die Suchrichtung beeinflusst, sondern leider auch die Suchgeschwindigkeit. Im Fall der Demoversion ist der beste Faktor 1.68f. Somit berechnet sich die Heuristik im Fall einer falschen Wegführung:

```
GkDouble dDistance = ((fXd * fXd) + (fYd * fYd)) * 1.68f
```

Dieser Faktor sorgt dafür, dass eine Suchrichtung, die das erste Portal ignoriert, nicht beachtet wird, weil die Gesamtentfernung in diesem Fall zu groß wird. Ignorieren bedeutet, dass die Gesamtentfernung so groß ist, dass dieser Knoten bei der Heap-Sortierung, eine schlechte Position jenseits der Wurzel belegt.

Nun muss nur noch beantwortet werden, wie man feststellen kann, ob der Pfad das nächste Portal durchkreuzt. Prinzipiell ist dies sehr einfach. Da jedes Portal durch zwei Punkte aufgespannt wird, ist es sehr leicht eine Geradengleichung für das Portal aufzustellen. Da jeder Suchpunkt, wie erwähnt, seinen Vater-/ Vorgängerknoten und die Steigung mit diesem kennt, ist es auch hier sehr leicht möglich, eine Geradengleichung aufzustellen und zu überprüfen, ob sich die Geraden schneiden. Wenn dies der Fall ist, dann muss nur noch überprüft werden, ob der Schnittpunkt im Bereich des Portals liegt. Erstaunlicherweise reicht es aus, wenn der Schnittpunkt im Bereich der x- Werte der beiden Portalaußenpunkte liegt. Im Gegenteil wird mitunter ein schlechterer Pfad erstellt, wenn die y- Werte mit einbezogen werden. Dies passiert vor allem bei großen Kurven. Liegt der Schnittpunkt im Bereich der x- Werte, dann wird kein Faktor verwendet:

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

GkDouble dDistance = (fXd * fXd) + (fYd * fYd)

Ansonsten schon:

GkDouble dDistance = ((fXd * fXd) + (fYd * fYd)) * 1.68f

<source code>

```
__forceinline const GkDouble GkCPathfindingSystem::GetHeuristic(GkCPathfindingNode &gkNode, const GkFloat
FloatGoalXPosition, const GkFloat FloatGoalYPosition)
{
	GkDouble dEstimatedHeuristic = 0;
	switch(m_ui08HeuristicType)
	{
		case HEURISTIC_TYPE_OVERESTIMATED:
		{
			// hard to find correct, realistic values
		}
		break;
		case HEURISTIC_TYPE_NORMAL:
		{
			if(m_ui08PortalToPortalSearch) // PPTP
				dEstimatedHeuristic = CostFromNodeToNode(gkNode, FloatGoalXPosition,
					FloatGoalYPosition); // Pytagoras
```

<Erläuterung>

Wenn m_ui08PortalToPortalSearch > 0 ist, dann findet eine normale A* Suche statt, d.h. hier wird nur der Satz des Pythagoras verwendet, um festzustellen, welche Entfernung die Kürzeste ist.

</Erläuterung>

```
			else
			{
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
                    dEstimatedHeuristic = CostFromNodeToNode(gkNode, FloatGoalXPosition,
                                                FloatGoalYPosition); // Pytagoras
```

<Erläuterung>

Auch hier wird erst einmal nur der Satz des Pythagoras verwendet.

</Erläuterung>

```
                    GkuInt08 ui08AddCostToNonStraightPath = AddCostToNonStraightPath(gkNode,
                                                FloatGoalXPosition, FloatGoalYPosition);
```

<Erläuterung>

In AddCostToNonStraightPath(..) wird die Steigung des Suchpunktes, die noch öfters notwendig ist, z.B. um herauszufinden, in welche Laufrichtung sich der Charakter während des Laufens drehen muss, erstellt.

</Erläuterung>

```
                    if(!IsIntersectingWithPortal(gkNode, FloatGoalXPosition, FloatGoalYPosition))
                        dEstimatedHeuristic *= 1.68;
```

<Erläuterung>

In IsIntersectingWithPortal(..) wird geprüft, ob sich die Gerade des Portals mit der Gerade des Suchknotens schneidet; falls nein, dann wird der Penaltyfaktor aufgerechnet.

</Erläuterung>

```
                    }
                }
                break;
                default:
                    dEstimatedHeuristic = 0;
        }

        return dEstimatedHeuristic;
}
```

</source code>

Prinzipiell funktioniert die Schnittpunktermittlung in IsIntersectingWithPortal(..) folgendermaßen:

Zuerst stellt man die Geradegleichung des Portals auf

$y = m_1 * x + b_1$,

m_1 = Steigung des Portals = GkCPortal::m_m (vorberechnet)

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

b_1 = y- Achsenabschnitt des Portals = GkCPortal::m_b (vorberechnet)

und die Geradengleichung des Suchpunktes

$y = m_2 * x + b_2$,

m_2 = Steigung des Suchpunktes = GkCPathfindingNode::m_dGradient (vorberechnet)

b_2 wird in IsIntersectingWithPortal(..) berechnet.

Zur Ermittlung des x- Schnittpunktes werden die Gleichungen gleichgesetzt und nach x aufgelöst:

$$m_1 * x + b_1 = m_2 * x + b_2$$

$$? \quad x = (b_2 - b_1) / (m_1 - m_2)$$

Dieser x- Wert, wird mit den Außenpunkten des Portals verglichen.

<source code>

```
__forceinline GkuInt08 GkCPathfindingSystem::IsIntersectingWithPortal(const GkCPathfindingNode &gkcCurrentNode, const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition)
{
	// note: bei Portalen können die beiden x-Werte niemals gleich sein
	if(m_ui08FindFinalGoal >= 2)
		return 1; // keine Portale vorhanden
	else
	{
		if(!gkcCurrentNode.GetNodeParent())
			return 0;

		if(!m_ui08FindFinalGoal)
		{
			if(!m_gkSPortalPath.gkCLastPortal->GetNodeParent())
				return 0;
		}

		if(!m_gkCWalkThroughPortal)
		{
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

<Erläuterung>

Ermittlung des Portals, das durchlaufen werden soll.

</Erläuterung>

```
                    m_gkCWalkThroughPortal =
                        m_gkCPortalBank->GetPortal(m_gkSPortalPath.gkCLastPortal->GetLastArea() - 1,
                        m_gkSPortalPath.gkCLastPortal->GetPoint().fXPos,
                         m_gkSPortalPath.gkCLastPortal->GetPoint().fYPos);
            }
        }

        if(!m_gkCWalkThroughPortal)
            return 0;


        GkDouble m = gkcCurrentNode.GetGradient();

        if(m == m_gkCWalkThroughPortal->GetGradient()) // gleiche Steigung
            return 0;
```

<Erläuterung>

Steigung des Portals und des Knotens werden verglichen, um eine Division durch 0 zu vermeiden.

</Erläuterung>

```
        // durch null teilen
        if((m - m_gkCWalkThroughPortal->GetGradient()) == 0)
            m += 0.00001;

        GkDouble b = gkcCurrentNode.GetNodeParent()->GetPoint().fYPos - (m * gkcCurrentNode.GetNodeParent()-
            >GetPoint().fXPos);
```

<Erläuterung>

Der y- Achsenabschnitt für den Knoten wird hier ermittelt.

</Erläuterung>

```
        GkDouble d32XTest = (m_gkCWalkThroughPortal->GetYAchsenAbschnitt() - b) / (m - m_gkCWalkThroughPortal-
            >GetGradient());
```

<Erläuterung>

d32Xtest ist der x- Wert des Schnittpunktes.

</Erläuterung>

```
        GkInt32 i32MinX = m_gkCWalkThroughPortal->GetMinX();
        GkInt32 i32MaxX = m_gkCWalkThroughPortal->GetMaxX();
        <Erläuterung>
        Ermittlung der zwei Außenpunkte des Portals.
        </Erläuterung>

        if(d32XTest < i32MinX || d32XTest > i32MaxX)
                return 0;
        <Erläuterung>
        Vergleich, ob x – Schnittpunkt auf Portal liegt, falls nein wird 0
        zurückgegeben.
        </Erläuterung>

        return 1;
}
</source code>
```

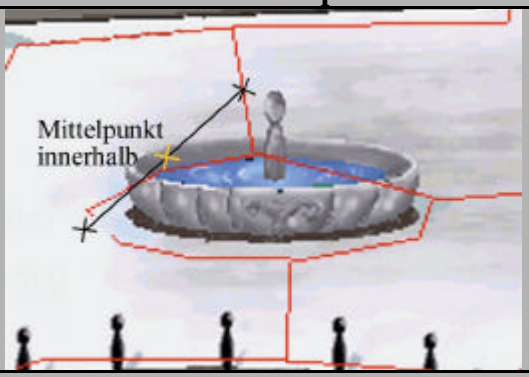
Ein Vorteil dieser Methode ist, dass kein *straight path* und *smoothing* mehr notwendig ist, weil der perfekte Pfad bereits erstellt wurde.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Last Portal Search (LPS)

Nach der PTP Suche befindet man sich eine Area, d.h. ein Portal, von der Zielarea entfernt. Ab der LPS finden wieder nur reine A* Suchen ohne Penaltyfaktor statt. Prinzipiell könnte man vom jetzigen Startpunkt aus direkt den Endpfad zum Zielpunkt errechnen. Aber die Heuristik ist eine äußerst starke Kraft und kann bei bestimmten Hindernissen, die nicht generalisierbar sind, urplötzlich einen Weg einschlagen, der nicht mehr der Kürzeste ist. Das Problem ist, dass in diesen Fällen die heuristische Abschätzung über das Hindernis dermaßen klein ist, dass eine andere Richtung, als die durch die PPTP geplante, gewählt wird, d.h. es wird auf ein anderes Portal zugesteuert, als vorgesehen. In diesem Fall würde die Suche vielleicht noch eine Area einschließen, die nicht mit der PPTP ermittelt wurde. Um dieser Problematik entgegenzuwirken ermittelt man nicht den Weg direkt zum Zielpunkt, sondern den Pfad zum Mittelpunkt der Strecke zwischen dem letzten Portalpunkt und dem Zielpunkt.

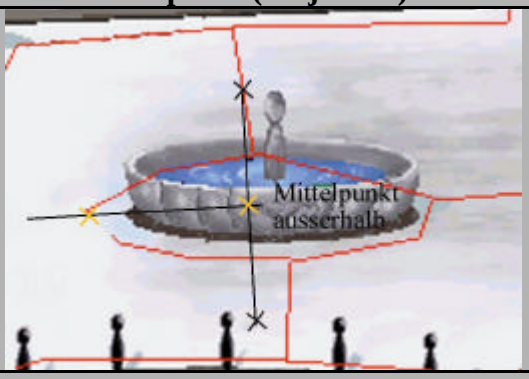
Abbildung 37:
Wahl des Mittelpunktes zwischen Portal und Zielpunkt



Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Sollte dieser Punkt nicht in der Zielarea liegen, dann wird er solange entlang der Mittelsenkrechen in die Area projiziert, bis ein legaler Punkt gefunden wird. Dieser ist dann der neue Zielpunkt. Wie auch bei der PTP wird diese Suche solange durchgeführt, bis die nächste Area, also die Zielarea erreicht ist.

Abbildung 38:
Mittelpunkte zwischen Portal und Zielpunkt (Projektion)



Die verwendetet Methode zur Ermittlung des Mittelpunktes ist CreateLegalNodeBetweenGoalAndPortal(..).

<source code>

```
__forceinline GkuInt08 GkCPathfindingSystem::CreateLegalNodeBetweenGoalAndPortal(const GkFloat FloatGoalXPosition,
const GkFloat FloatGoalYPosition, GkPoint &gkPoint)
{
	if(!m_gkSPortalPath.gkCCurrentPortal)
		return 0;

	// try to find legal node on (half of) the line between portal and goal
	GkFloat fDeltaX = ((FloatGoalXPosition - m_gkSPortalPath.gkCCurrentPortal->GetPoint().fXPos) / 2);
	GkFloat fDeltaY = ((FloatGoalYPosition - m_gkSPortalPath.gkCCurrentPortal->GetPoint().fYPos) / 2);
	GkFloat fNewNodeOnHalfX = m_gkSPortalPath.gkCCurrentPortal->GetPoint().fXPos + fDeltaX;
	GkFloat fNewNodeOnHalfY = m_gkSPortalPath.gkCCurrentPortal->GetPoint().fYPos + fDeltaY;

	// same area ??
	GkuInt08 ui08AreaGoal = m_WorldKnowledge->GetFloorCoveringType(FloatGoalXPosition, FloatGoalYPosition);
	if(!ui08AreaGoal)
		return 0; // goal cannot be reached

	GkuInt08 ui08AreaHalfNode = m_WorldKnowledge->GetFloorCoveringType((GkInt32)fNewNodeOnHalfX,
		(GkInt32)fNewNodeOnHalfY);
	if(ui08AreaHalfNode && (ui08AreaGoal == ui08AreaHalfNode))
```

```
{
        gkPoint.fXPos = (GkInt32)fNewNodeOnHalfX;
        gkPoint.fYPos = (GkInt32)fNewNodeOnHalfY;
        return 1;
}

// search one direction
GkDouble m = 0;
if(!fDeltaX)
        m = (fDeltaY + 0.1) / (0.1);
else
        m = fDeltaY / fDeltaX;

// Kehrwert
if(!m)
        return 0; // use goal (zu kleine Wahrscheinlichkeit)
else
        m = -1/m;

GkDouble b = fNewNodeOnHalfY - (m * fNewNodeOnHalfX);

// choose direction
GkuInt08 ui08Direction = 0;
while(ui08Direction < 2)
{
        GkFloat fCheckX = fNewNodeOnHalfX;
        GkFloat fCheckY = fNewNodeOnHalfY;

        GkInt08 i08Add = -1;
        if(ui08Direction)
                i08Add = 1;

        GkuInt08 ui08Area = 0;

        while(!ui08Area)
        {
                fCheckX += i08Add;
                fCheckY = (m * fCheckX) + b;

                ui08Area = m_WorldKnowledge->GetFloorCoveringType((GkInt32)fCheckX,
                        (GkInt32)fCheckY);

                if(ui08Area && (ui08Area != ui08AreaGoal))
                        break;
                else if(ui08Area)
                {
                        gkPoint.fXPos = (GkInt32)fCheckX;
                        gkPoint.fYPos = (GkInt32)fCheckY;
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
                                GkFloat fCheckX10 = fCheckX + (i08Add * 2);
                                GkFloat fCheckY10 = (m * fCheckX10) + b;

                                GkuInt08 ui08Area10 = m_WorldKnowledge-
                                        >GetFloorCoveringType((GkInt32)fCheckX10, (GkInt32)fCheckY10);
                                if(ui08Area10 && (ui08Area10 == ui08AreaGoal))
                                {
                                        gkPoint.fXPos = (GkInt32)fCheckX10;
                                        gkPoint.fYPos = (GkInt32)fCheckY10;
                                }
                                return 1;
                        }
                        else if((fCheckX < 1) || (fCheckX > 799))
                                break;
                        else if((fCheckY < 1) || (fCheckY > 599))
                                break;
                }

                ui08Direction++;
        }


        return 0;
}
```

</source code>

Goal Area Search (GAS)

Bei der GAS findet, abgesehen von der PPTP, als einziges eine reine A* Suche statt, die nicht von außen abgebrochen wird. Normalerweise wird die Suche abgebrochen, wenn der erste Punkt der nächsten Area erreicht wird. Nachdem der Zielpunkt erreicht wurde, wird der Gesamtpfad zusammengesetzt.

Direkte Pfade

Aufgrund der Tatsache, dass die A* Suche sehr umfangreich ist, wird vor jeder Suche versucht, einen direkten Pfad zwischen den jeweiligen Start- und Zielpunkten zu ermitteln. Dies gilt natürlich nicht für die PPTP. Der angenehme Nebeneffekt ist, dass durch diese Vorgehensweise noch ästhetischere, natürlichere Pfade entstehen.

Fazit

Die *forced portal A** Suche ist die einzige Methode, die perfekte, auf die Ästhetik ausgerichtete Pfade generiert, sodass der Eindruck entsteht, dass ein Mensch tatsächlich einen solchen Weg gehen würde. Sie hat noch einen weiteren großen Vorteil. Die Demoversion betrachtet das Pathfinding nur unter dem Aspekt einen Pfad anhand von festen Hindernissen zu generieren. Sollte man jetzt anstreben ein Kollisionssystem[79] zu implementieren, dann wäre dies sehr leicht umsetzbar, weil bei einer Kollision nur der Weg vom momentanen Kollisionspunkt in die nächste, vielleicht in die übernächste Area generiert werden müsste. Der Rest des Pfades bräuchte nicht, wie es oftmals gemacht wird, verworfen werden. Dies hätte zur Folge, dass die Performanz sehr stark entlastet wird.

[79] *Kollisionssystem*: Dieses System regelt, dass zwei Charaktere nicht zusammenstoßen oder durcheinander laufen

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Pathfindingsystem

*Implementierung des A**

GkCPathfindingsystem

An dieser Stelle soll die Deklaration der Klasse GkCPathfindingsystem eingefügt werden, um einen prinzipiellen Eindruck zu erhalten, welche Methoden die Klasse beinhaltet. Die für die *forced Portal A** Suche wichtigsten Methoden wurden bereits erläutert. Prinzipiell regelt GkCPathfindingsystem die gesamte Suche, beginnend beim PPTP, über PTP, über LAS, über GAS. Welche Suche wann durchgeführt wird, wird in Findpath(..) bestimmt. Alle anderen Methoden beziehen sich grundsätzlich auf die Suche.

<source code>

```
class GkCPathfindingSystem
{
public: // Kon-/ Destruktor
        GkCPathfindingSystem();
        ~GkCPathfindingSystem();

public:
        // A* Pathfinding method
        GkInt08 FindPath(const GkFloat FloatStartXPosition, const GkFloat FloatStartYPosition, const GkFloat
                FloatGoalXPosition, const GkFloat FloatGoalYPosition);;
        // Setup of World Knowledge
        void SetupWorldKnowledge(const GkCLocationGrid *WorldKnowledge, GkCVector<GkCPathfindingNode>
                *gkCNodeBankVector, const GkCPortalManager *gkCPortalBank, const GkSSpecialAreas *gkSNoAreas =
                NULL, const GkSSpecialAreas *gkSPreferedAreas = NULL);
        // set different heruristics
        void SetHeuristicType(const GkuInt08 ui08HeuristicType);
        // draw path
        void Draw(GkuInt08 ui08Path);
        // return Destination
        const GkPoint& GetDestination() const;
        // current movement point & area
        const GkSMovInfo* GetNextWayInfo();
        // same as GetNextWayInfo, but with decreasing iterator
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
        const GkSMovInfo* CheckNextWayInfo();
        // get amount of path nodes
        const GkuInt32 GetNodes() const;
        // get search time
        const GkDouble GetSearchTime() const;

protected:
        // stop search ??
        GkInt08 StopAStarSearch(GkCPathfindingNode& gkCBestNode, const GkFloat FloatGoalXPosition, const GkFloat
FloatGoalYPosition, const GkFloat fGoalXPosition_, const GkFloat fGoalYPosition_);
        // is there a direct way
        GkuInt08 FindDirectPath(const GkFloat FloatCurrentXPosition, const GkFloat FloatCurrentYPosition, const GkFloat
FloatGoalXPosition, const GkFloat FloatGoalYPosition, GkCPathfindingNode* gkCParent, GkuInt08 ui08CheckArea = 0);
        // individual heruritisc funtion
        const GkDouble GetHeuristic(GkCPathfindingNode &gkNode, const GkFloat FloatGoalXPosition, const GkFloat
FloatGoalYPosition);
        // Initializing/ Cleaning Open and Close List
        void InitializeOpenCloseList();
        // returns new node from Nodebank, if no node with this Position is found
        // else gets node from Masternode list (Closed or Open Lis)
        GkCPathfindingNode *GetNode(const GkFloat FloatStartXPosition, const GkFloat FloatStartYPosition);
        // returns new node from Nodebank, if no node with this Position is found
        // else gets node from Masternode list (Closed or Open Lis)
        // does not check if node is possible
        GkCPathfindingNode *GetNewNode(const GkFloat FloatStartXPosition, const GkFloat FloatStartYPosition);
        // actual A* search
        GkInt08 DoAStarSearch(const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition, const GkFloat
fGoalXPosition_, const GkFloat fGoalYPosition_, const GkFloat fXStart, const GkFloat fYStart, GkuInt08 ui08CheckStart);
        // Creates array of successor of current node
        void GetSuccessors(const GkFloat FloatCurrentXPosition, const GkFloat FloatCurrentYPosition, const GkFloat
FloatGoalXPosition, const GkFloat FloatGoalYPosition);
        // calculates distance between two nodes
        GkDouble CostFromNodeToNode(const GkCPathfindingNode &gkcFirstNode, const GkCPathfindingNode
&gkcSecondNode);
        GkDouble CostFromNodeToNode(const GkCPathfindingNode &gkcFirstNode, const GkFloat fSecondXPosition,
const GkFloat fSecondYPosition);
        // search abortion
        GkuInt08 SearchAbortion(const GkCPathfindingNode &gkcCurrentNode);
        // add cost to not straight path
        GkDouble AddCostToNonStraightPath(GkCPathfindingNode &gkcCurrentNode, const GkFloat FloatGoalXPosition,
const GkFloat FloatGoalYPosition);
        // end pathfinding
        GkuInt08 ClosestNodeToGoal(const GkFloat FloatStartXPosition, const GkFloat FloatStartYPosition, const GkFloat
FloatGoalXPosition, const GkFloat FloatGoalYPosition);
        GkuInt08 IsClosestNodeToGoal(const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition);
        // add cost to avoid choosing a shortest path over big obstacles
        //GkuInt32 AddCostToHeuristicOverObstacle(const GkCPathfindingNode &gkcCurrentNode, const GkFloat
FloatGoalXPosition, const GkFloat FloatGoalYPosition);
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
        //
        GkuInt08 IsIntersectingWithPortal(const GkCPathfindingNode &gkcCurrentNode, const GkFloat FloatGoalXPosition,
const GkFloat FloatGoalYPosition);
        // pathfind shortest way through areas
        GkuInt08 CreatePortalToPortalPath(const GkFloat FloatStartXPosition, const GkFloat FloatStartYPosition);
        // create portal path
        void CreatePortalPath();
        // CreateNonPortalGoalValues
        GkuInt08 CreateNonPortalGoalValues(const GkFloat FloatStartXPosition, const GkFloat FloatStartYPosition);
        // Create Start node
        GkuInt08 CreateStartNode(const GkFloat fNewStartX, const GkFloat fNewStartY, const GkFloat fGoalX, const
GkFloat fGoalY, const GkuInt08 ui08Reset = true);
        // CreateLegalNodeBetweenGoalAndPortal
        GkuInt08 CreateLegalNodeBetweenGoalAndPortal(const GkFloat FloatGoalXPosition, const GkFloat
FloatGoalYPosition, GkPoint &gkPoint);
        // smooth path
        GkuInt08 DoSmoothing();
        // Add Smoothing Nodes by using catmull rom formula
        void AddSmoothingNodes(GkCPathfindingNode &gkCNodeOne, GkCPathfindingNode &gkCNodeTwo,
GkCPathfindingNode &gkCNodeThree, GkCPathfindingNode &gkCNodeFour, GkCPathfindingNode &gkChildOfTwo);
        // donnot walk on this
        GkuInt08 IsAreaAllowed(GkuInt08 ui08Area);
        // reset open list
        void ResetOpenList();
        // prefere area ??
        GkuInt08 ShouldAreaBePrefer(GkuInt08 ui08Area);
        // node list ok?
        GkuInt08 DoErrorCheck(const GkFloat fNewStartX, const GkFloat fNewStartY, GkuInt08 ui08New = 0);

private:
        const GkCLocationGrid *m_WorldKnowledge;      // WorldNodeData;
        const GkCPortalManager *m_gkCPortalBank;              // PortalBank
        GkuInt08 m_ui08HeuristicType;                         // type of the heuristic: no heuristic, normal,
overestimated
        GkCHeapPriorityQueue<GkCPathfindingNode*> m_gkCOpenHeapPriorityQueue; // Open list heap priority queue
        GkCPathfindingNode *m_gkCSuccessors;          // actual point successor
        GkC2DPtHashTable<GkCPathfindingNode*, GkPoint> *m_MasterNodeHashTable; // OPen & close list hash table for
fast search
        GkCVector<GkCPathfindingNode> *m_gkCNodeBankVector; // memory for a* search (one bank for everybody
        GkCPathfindingNode* m_gkCGoalNode;                    // Final node, is not GoalPoint if search is aborted
        GkCPathfindingNode* m_gkCNextNode;                    // How far did the actor walk
        GkuInt32 m_ui32Time;                                                  // time needed for the search
        GkuInt32 m_uiAbortionTime;                                    // time when search should be aborted
        GkuInt08 m_ui08MaxSuccessors;                                 // Max Successors from current control
point
        GkuInt08 m_ui08EveryOtherNode;                                // check just every other successor, [0,1]
        GkuInt08 m_ui08CheckSearchAbortion;                   // should the search be aborted
        GkCLocationGrid *m_gkCDetectedArea; // Suchausbreitungsbereichbreich
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
        GkFloat m_fGoalX; // Goal X Point
        GkFloat m_fGoalY; // Goal Y Point
        GkuInt08 m_ui08CheckDirectPath;                    // should direct path be checked
        GkuInt08 m_ui08StopObstacleCheck;
        GkuInt08 m_ui08PortalToPortalSearch;
        GkuInt08 m_ui08Smoothing;
        GkInt32 m_i32Check;
        const GkSSpecialAreas *m_gkSNoAreas; // Areas on which agent is not allowed to walk
        const GkSSpecialAreas *m_gkSPreferedAreas; // prefered areas
        GkuInt08 m_ui08FindFinalGoal; // 0: find way between portals;
                                                                    // 1: find way through last portal to final
area
                                                                    // 2: do normal astar without portals
                                                                    // 3: find way in area
        GkCPortal *m_gkCWalkThroughPortal; // go through this portal

        struct GkSPortalPath
        {
                GkCPathfindingNode *gkCCurrentPortal;
                GkCPathfindingNode *gkCLastPortal;
                GkCPathfindingNode *gkCPathfindingNode;
                GkuInt08 ui08Iterator;
                GkuInt08 ui08MaxNodes;
        }m_gkSPortalPath;

        GkPoint m_gkDestination;
        struct GkSPath
        {
                GkSMovInfo *gkSMovInfo;
                GkuInt32 ui32Iterator;
                GkuInt32 ui32MaxIterator;
                GkuInt32 ui32MaxNodes;
        }m_gkSPath;
};
```

</source code>

Die Klasse GkCPathfindingsystem regelt, wie der Name schon sagt, die Pfadfindung. Bestandteil dieser Klasse ist die Methode DoAStarSearch(..), die die Umsetzung des A*- Algorithmus darstellt.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

A*- Algorithmus

Um die kleinen Unterschiede zwischen dem Pseudocode des A* und der Implementierung zu erläutern, wird hier auf die Implementierung eingegangen.

<source code>

```
__forceinline GkInt08 GkCPathfindingSystem::DoAStarSearch(const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition, const GkFloat fGoalXPosition_, const GkFloat fGoalYPosition_, const GkFloat fXStart, const GkFloat fYStart, GkuInt08 ui08CheckStart)
{
```

<Erläuterung>

Folgende Werte werden dem A* übergeben:

- const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition:
  Zielpunkt der momentanen Suche,
- const GkFloat fGoalXPosition_, const GkFloat fGoalYPosition_ :
  tatsächlicher Zielpunkt der Gesamtsuche,
- const GkFloat fXStart, const GkFloat fYStart:
  tatsächlicher Startpunkt der Gesamtsuche,
- GkuInt08 ui08CheckStart:
  besagt, ob DoAStarSearch(..) das erste mal aufgerufen wird.

</Erläuterung>

```
    m_gkCWalkThroughPortal = NULL;
```

<Erläuterung>

Wie festgestellt wurde, wird die Suche beim PTP anhand von Portalen durchgeführt. m_gkCWalkThroughPortal ist das nächste Portal, durch das ein Weg zum Übernächsten gefunden werden muss. m_gkCWalkThroughPortal muss hier resetet werden, um in IsIntersectingWithPortal(..) neu erstellt zu werden.

</Erläuterung>

```
    while(!m_gkCOpenHeapPriorityQueue.IsEmpty())
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

<Erläuterung>

Solange noch ein Element in der Open List ist, muss die A* Suche fortgesetzt werden. Sollte kein Element mehr vorhanden sein, dann gibt es keinen Weg von A nach B und die Suche ist gescheitert.

</Erläuterung>

```
{
        // get most admiring node
        GkCPathfindingNode* gkCBestNode = m_gkCOpenHeapPriorityQueue.Pop();
```

<Erläuterung>

Hier wird der Knoten mit der kürzesten Entfernung zum Zielpunkt aus der Open List entfernt.

</Erläuterung>

```
        if(m_ui08FindFinalGoal == 3) // don't leave this area
```

<Erläuterung>

m_ui08FindFinalGoal == 3 bedeutet, man befindet sich in der GAS Phase. Hier wird dafür gesorgt, dass die Zielarea nicht mehr verlassen wird.

</Erläuterung>

```
        {
                if(m_WorldKnowledge->GetFloorCoveringType(gkCBestNode->GetPoint().fXPos,
                        gkCBestNode->GetPoint().fYPos) != m_WorldKnowledge-
                        >GetFloorCoveringType(FloatGoalXPosition, FloatGoalYPosition))
                        continue;
        }

        if(ui08CheckStart)
        {
                if((gkCBestNode->GetPoint().fXPos == fXStart) && (gkCBestNode->GetPoint().fYPos ==
                        fYStart))
                        gkCBestNode->SetNodeParent(NULL);
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

<Erläuterung>

Hier wird dafür gesorgt, wenn das erste Mal die PTP durchgeführt wird, dass der Startknoten keinen Vorgänger/ Vaterknoten hat, um zu vermeiden, dass sich die Suche im Kreis dreht.

</Erläuterung>

```
}
// ~neu 09.01.02


if(StopAStarSearch(*gkCBestNode, FloatGoalXPosition, FloatGoalYPosition, fGoalXPosition_,
        fGoalYPosition_))
```

<Erläuterung>

Tritt dieser Fall ein, dann wurde ein korrekter Pfad gefunden.

</Erläuterung>

```
        return 1;
else if(!m_ui08FindFinalGoal && !m_ui08PortalToPortalSearch)
{
        if(m_gkSPortalPath.gkCCurrentPortal)
        {
                if(m_gkSPortalPath.ui08Iterator && m_gkSPortalPath.ui08MaxNodes)
                {
                        if(m_WorldKnowledge->GetFloorCoveringType(gkCBestNode-
                                >GetPoint().fXPos, gkCBestNode-
                                >GetPoint().fYPos) !=
                        m_gkSPortalPath.gkCPathfindingNode[m_gkSPortalPath.ui08Iterator–
                        1].GetLastArea())
                        {
                                if(m_WorldKnowledge-
                                >GetFloorCoveringType(gkCBestNode->GetPoint().fXPos,
                                gkCBestNode->GetPoint().fYPos) !=
                                m_gkSPortalPath.gkCCurrentPortal->GetLastArea())
```

<Erläuterung>

Hier wird vermieden, dass man in die Area zurückkehrt, aus der man gerade gekommen ist.

</Erläuterung>

```
                                        continue;
                        }
                }
                else if(m_gkSPortalPath.ui08MaxNodes)
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
                    {
                        if(gkCBestNode->GetNodeParent())
                        {
                            if(m_WorldKnowledge->GetFloorCoveringType(gkCBestNode->GetPoint().fXPos, gkCBestNode->GetPoint().fYPos) != m_WorldKnowledge->GetFloorCoveringType(gkCBestNode->GetNodeParent()->GetPoint().fXPos, gkCBestNode->GetNodeParent()->GetPoint().fYPos))
                            {
                                if(m_WorldKnowledge->GetFloorCoveringType(gkCBestNode->GetPoint().fXPos, gkCBestNode->GetPoint().fYPos) != m_gkSPortalPath.gkCCurrentPortal->GetLastArea())
                                <Erläuterung>
                                Auch hier wird vermieden, dass man in die Area zurückgeht, aus der man gerade gekommen ist.
                                </Erläuterung>

                                    continue;
                            }
                        }
                    }
                }
            }
        }

        // Get successors of best node
        GkuInt08 ui08Counter = m_ui08MaxSuccessors;
        <Erläuterung>
        Hier werden die Nachbarknoten ermittelt. Die Methode unterscheidet, ob man sich in der PPTP Phase befindet oder nicht.
        </Erläuterung>
        GetSuccessors(gkCBestNode->GetPoint().fXPos, gkCBestNode->GetPoint().fYPos, FloatGoalXPosition, FloatGoalYPosition);
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
// check each successor
while(ui08Counter--)
{
```

<Erläuterung>

Hier wird für jeden Nachbar-/ Kindknoten die A* Suche durchgeführt.

</Erläuterung>

```
// is successor represented in the world data
if(!m_ui08PortalToPortalSearch)
{
        if(!m_WorldKnowledge-
        >IsSolidFloorCoveringType(m_gkCSuccessors[ui08Counter].GetPoint().fXPos,
        m_gkCSuccessors[ui08Counter].GetPoint().fYPos))
                continue;
        else if(ui08CheckStart)
        {
                if((m_gkCSuccessors[ui08Counter].GetPoint().fXPos == fXStart) &&
                (m_gkCSuccessors[ui08Counter].GetPoint().fYPos == fYStart))
                {
                        m_gkCSuccessors[ui08Counter].SetNodeParent(NULL);
                        continue;
                }
        }
}
else
{
        if(m_gkCSuccessors[ui08Counter].GetPoint().fXPos == -1)
                continue;
}
```

<Erläuterung>

Führe Vergleich nur durch, wenn dieser Punkt nicht der vorherige ist.

</Erläuterung>

```
// don't search the node we just came from
if((!gkCBestNode->GetNodeParent()) ||
  (gkCBestNode->GetNodeParent()->GetPoint().fXPos !=
          m_gkCSuccessors[ui08Counter].GetPoint().fXPos ||
   gkCBestNode->GetNodeParent()->GetPoint().fYPos!=
m_gkCSuccessors[ui08Counter].GetPoint().fYPos))
{
        m_gkCSuccessors[ui08Counter].SetNodeParent(gkCBestNode);
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
GkDouble dCostNodeToNode =
CostFromNodeToNode(m_gkCSuccessors[ui08Counter], *gkCBestNode);
m_gkCSuccessors[ui08Counter].SetCostFromStart(gkCBestNode-
>GetCostFromStart() + dCostNodeToNode);

m_gkCSuccessors[ui08Counter].SetTotalCost(m_gkCSuccessors[ui08Counter].GetCostFromSta
rt() + GetHeuristic(m_gkCSuccessors[ui08Counter], FloatGoalXPosition, FloatGoalYPosition));
```

<Erläuterung>

Besorge dir das Adäquat zu diesem Knoten, aus der Hash Table.

</Erläuterung>

```
// get actual node which is the same as m_gkCSuccessors[ui08Counter],
// but the search for now is faster in the hash table
GkCPathfindingNode* gkCActualNode =
GetNode(m_gkCSuccessors[ui08Counter].GetPoint().fXPos,
m_gkCSuccessors[ui08Counter].GetPoint().fYPos);
```

<Erläuterung>

Wenn der neue Knoten schon in der Open oder Closed List ist und eine bessere Gesamtentfernung hat, dann setzt sich die Suche fort. Ist dies nicht der Fall, dann gibt es einen kürzeren Weg über diesen Knoten, der der Open List hinzugefügt oder in ihr aktualisiert werden muss.

</Erläuterung>

```
// the following only takes: time = O(1)
if((gkCActualNode->IsOnOpen() || gkCActualNode->IsOnClosed())
        && m_gkCSuccessors[ui08Counter].GetTotalCost() >= gkCActualNode-
        >GetTotalCost())
        continue;
else
{
        // node seams to be promising
        // push it onto open
        gkCActualNode->SetOnClosed(0);
        gkCActualNode->SetNodeParent(gkCBestNode);
        gkCActualNode-
        >SetCostFromStart(m_gkCSuccessors[ui08Counter].GetCostFromStart());
        gkCActualNode-
        >SetTotalCost(m_gkCSuccessors[ui08Counter].GetTotalCost());
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
                                    gkCActualNode-
                                        >SetLastArea(m_gkCSuccessors[ui08Counter].GetLastArea());
                                    gkCActualNode-
                                        >SetGradient(m_gkCSuccessors[ui08Counter].GetGradient());

                                    if(gkCActualNode->IsOnOpen())
                                        m_gkCOpenHeapPriorityQueue.Update(gkCActualNode);
                                    else
                                    {
                                        gkCActualNode->SetOnOpen(1);
                                        m_gkCOpenHeapPriorityQueue.Push(gkCActualNode);
                                    }


                                    // search too long
                                    <Erläuterung>
                                    Dauert die Suche zu lange, wird sie abgebrochen.
                                    </ Erläuterung>
                                    if(SearchAbortion(*gkCBestNode))
                                        return 0;


                                    <Erläuterung>
                                    Geprüfter Punkt/ Knoten muss markiert werden,
                                    um ihn später blau als Suchmenge auszugeben.
                                    </Erläuterung>
                                    m_gkCDetectedArea->SetFloorCoveringTypeAsLegal(gkCActualNode-
                                        >GetPoint().fXPos, gkCActualNode->GetPoint().fYPos);
                                }
                            }
                        }
                        <Erläuterung>
                        Füge den Vaterknoten der Closed List hinzu.
                        </Erläuterung>
                        // best was explored; push onto closed
                        gkCBestNode->SetOnClosed(1);
                }

                return 0;
        }
        </source code>
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Man sieht, dass sich, bis auf einzelne Nuancen, die die Suche ein wenig schneller gestalten oder die spezifisch für die *forced portal A** Suche sind, die Implementierung nur leicht vom Pseudocode unterscheidet.

Die phänomenale Gestalt – Teil 2

Motivation

Im Beispielprogramm wird gezeigt, wie leicht mittels der *forced portal A** Suche eine Beeinflussung hinsichtlich der Motivation ist.

Es gibt drei Charaktere, die sich in der Lokation „Stadt“, in Bezug auf das Pathfinding, vollkommen unterschiedlich verhalten. Ein normaler, lebenslustiger Erwachsener (Gabriel) nimmt immer den kürzesten Weg. Für ihn spielt es keine Rolle, ob er einen normalen Weg nimmt oder über Brücken oder Eis läuft. Ein Kind (Avron) läuft lieber, weil es ein viel größeres Abenteuer ist, über die Eisschollen, als über die normalen Wege und Brücken. Eine dritte Person (Stuart), die schon einmal als Kind im Eis eingebrochen ist und seitdem eine Psychose bezüglich Eis und Wasser hat, betritt niemals das Eis.

Prinzipiell geht es bei der Motivation darum, entweder bestimmte Areas bei der Suche vorzuziehen oder zu ignorieren, d.h. nicht zu betreten. Aufgrund der Tatsache, dass bei der *forced portal A** Suche keine anderen Areas betreten werden, als die durch die PPTP Ermittelten, ist es sehr einfach, auf dieses System aufbauend, eine Motivation zu implementieren.

Am einfachsten ist dies bei dem Versuch bestimmte Areas nicht zu betreten. In diesem Fall werden die Portale bei der PPTP, die zu einer Area gehören, die nicht betreten werden dürfen, in der Methode GetSuccessors(..), einfach ignoriert. Somit werden diese Areas niemals betreten. Im Beispielprogramm ignoriert Stuart alle

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Eisflächen. Es wurde festgestellt, dass mit der Methode GkCAgent::SetupWorldKnowledge(..) für die Motivation die Areanummern übergeben werden, die ein bestimmter Agent nicht betreten darf (gkSNoAreas) und die Areanummern, die er bevorzugen soll (gkSPreferedAreas). Somit ist nur noch eine simple Abfrage bei der Auswahl der Portale notwendig, ob eine Area betreten werden darf . Hierfür gibt es folgende Methode:

<source code>
```
__forceinline GkuInt08 GkCPathfindingSystem::IsAreaAllowed(GkuInt08 ui08Area)
{
	if(m_gkSNoAreas)
	{
		for(GkuInt08 ui08Counter = 0; ui08Counter < m_gkSNoAreas->ui08MaxAreas; ui08Counter++)
		{
			if(m_gkSNoAreas->ui08Area[ui08Counter] == ui08Area)
				return 0;
		}
	}

	return 1;
}
```
</source code>

Die Abfrage, ob eine bestimmte Area bevorzugt werden sollte, wird ebenfalls während des PPTP geregelt. Prinzipiell gibt es auch hierfür eine Methode:

<source code>
```
__forceinline GkuInt08 GkCPathfindingSystem::ShouldAreaBePrefer(GkuInt08 ui08Area)
{
	if(m_gkSPreferedAreas)
	{
		for(GkuInt08 ui08Counter = 0; ui08Counter < m_gkSPreferedAreas->ui08MaxAreas; ui08Counter++)
		{
			if(m_gkSPreferedAreas->ui08Area[ui08Counter] == ui08Area)
				return 1;
		}
	}
	else
		return 1;

	return 0;
}
```
</source code>

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

</source code>

Die Entscheidung, ob eine Area bevorzugt werden soll (Avron bevorzugt die Eisflächen) wird, wie bei den verbotenen Areas, in GetSuccessors(..) getroffen. Die Umsetzung ist ebenfalls sehr einfach. Wenn ein Portal ermittelt wird, dass sich auf einer bevorzugten Area befindet, dann werden einfach alle weiteren Portale, bei denen das nicht der Fall ist, ignoriert, sodass ein bestimmtes Portal nur diesen einen Nachfolger hat. Somit ist garantiert, dass immer nur dieser Punkt für die Ermittlung eines kürzesten Pfades verwendet wird.

<source code>

```
__forceinline void GkCPathfindingSystem::GetSuccessors(const GkFloat FloatCurrentXPosition, const GkFloat FloatCurrentYPosition, const GkFloat FloatGoalXPosition, const GkFloat FloatGoalYPosition)
{
	if(m_ui08PortalToPortalSearch)
	{
		GkPoint gkPoint;

		// above
		gkPoint.fXPos = -1;
		gkPoint.fYPos = 0;
		m_gkCSuccessors[0].SetPoint(gkPoint);

		// beyond
		m_gkCSuccessors[1].SetPoint(gkPoint);

		// left
		m_gkCSuccessors[2].SetPoint(gkPoint);

		// right
		m_gkCSuccessors[3].SetPoint(gkPoint);

		// beyond left
		m_gkCSuccessors[4].SetPoint(gkPoint);

		// beyond right
		m_gkCSuccessors[5].SetPoint(gkPoint);

		// above left
		m_gkCSuccessors[6].SetPoint(gkPoint);

		// above right
		m_gkCSuccessors[7].SetPoint(gkPoint);
```

<Erläuterung>

Bei allen Nachbarknoten werden alle x- Werte auf –1 gesetzt, um später eine Abfragemöglichkeit zu haben, ob die Knoten untersucht werden sollen. Sie werden nur dann untersucht, wenn x ungleich -1 ist.

</Erläuterung>

```
GkuInt08 ui08Area = m_WorldKnowledge->GetFloorCoveringType(FloatCurrentXPosition,
                    FloatCurrentYPosition);
GkuInt08 ui08Parent = 0;
GkuInt08 ui08AreaParent = 0;
if(GetNode(FloatCurrentXPosition, FloatCurrentYPosition)->GetNodeParent())
{
        GkPoint gkParentLoc;
        gkParentLoc.fXPos = GetNode(FloatCurrentXPosition, FloatCurrentYPosition)-
        >GetNodeParent()->GetPoint().fXPos;
        gkParentLoc.fYPos = GetNode(FloatCurrentXPosition, FloatCurrentYPosition)-
        >GetNodeParent()->GetPoint().fYPos;

        ui08AreaParent = GetNode(FloatCurrentXPosition, FloatCurrentYPosition)->GetNodeParent()-
        >GetLastArea();
        if(ui08AreaParent == ui08Area)
                ui08Area = m_gkCPortalBank->GetOtherArea(ui08Area - 1, FloatCurrentXPosition,
        FloatCurrentYPosition); // 04.01.02
}

GetNode(FloatCurrentXPosition, FloatCurrentYPosition)->SetLastArea(ui08Area);
```

<Erläuterung>

Der momentane Suchpunkt wurde initialisiert und es wurde festgestellt, in welcher Area er sich befindet.

</Erläuterung>

```
if(ui08Area && IsAreaAllowed(ui08Area))
```

<Erläuterung>

Hier wird ermittelt, ob der Suchpunkt überhaupt gültig ist, dies ist der Fall, wenn ui08Area >0 ist, d.h. er ist ein Portalpunkt. Danach wird abgefragt, ob die Area des Punktes überhaupt betreten werden darf oder ob die Area für diesen Agenten gesperrt ist.

</Erläuterung>

```
{
        const GkSArea *gkSArea = m_gkCPortalBank->GetArea(ui08Area - 1);
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
if(!gkSArea || !gkSArea->uiIterator)
        return;

GkInt08 i08BestAreaPoint = -1;
for(GkuInt32 ui32Counter = 0; ui32Counter < gkSArea->uiIterator; ui32Counter++)
{
```

<Erläuterung>

Hier werden alle angrenzenden Portalmittelpunkte ermittelt.

</Erläuterung>

```
        gkSArea->gkCPortal[ui32Counter]->ChoosePointMiddle(FloatGoalXPosition,
        FloatGoalYPosition);
        const GkPoint *gkMiddleOfPortal = &gkSArea->gkCPortal[ui32Counter]-
                >GetPointMiddle();
        gkPoint.fXPos = gkMiddleOfPortal->fXPos;
        gkPoint.fYPos = gkMiddleOfPortal->fYPos;

        m_gkCSuccessors[ui32Counter].SetPoint(gkPoint);

        if(m_gkSPreferedAreas)
        {
                if(!ShouldAreaBePrefer(ui08Area))
                {
                        if(!ui08AreaParent || !ShouldAreaBePrefer(ui08AreaParent))
                        {
                                GkuInt08 ui08OArea = gkSArea-
                                >gkCPortal[ui32Counter]-
                                >GetOtherArea(ui08Area);
                                if(ui08OArea)
                                {
                                        if(ShouldAreaBePrefer(ui08OArea))
```

<Erläuterung>

Hier wird festgestellt, ob eine Area/ Portal bevorzugt werden sollte; falls ja, werden die x- Werte aller anderen Portale auf –1 gesetzt und somit nicht verwendet.

</Erläuterung>

```
                                        {
                                                i08BestAreaPoint =
```

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

```
                                                                    ui32Counter;
                                                                for(ui32Counter = 0;
                                                                ui32Counter <
                                                                i08BestAreaPoint;
                                                                ui32Counter++)
                                                                {
                                                                    gkPoint.fXPos = -
                                                                        1;
                                                                    gkPoint.fYPos = 0;
                                                            m_gkCSuccessors[ui32Counter].SetPoint(gkPoint);
                                                                }

                                                                break;
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                            else
...
}
```

</source code>

Eine Motivation kann nicht einfacher, für alle Charaktere individueller und eleganter gestalten werden. Jedem Charakter kann einfach die Liste der zu bevorzugenden oder zu vermeidenden Areas übermittelt werden und sie werden bevorzugt oder ignoriert. Ein großer Vorteil dieser Methode ist, dass keine weitere umfangreiche AI implementiert werden muss. Im source code werden gerade mal drei Abfragen für die Vermeidung und drei Abfragen für die Bevorzugung von Areas implementiert. Die Abfragen werden nur in der PPTP verwendet und beeinflussen somit nicht die Performanz der Suche.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

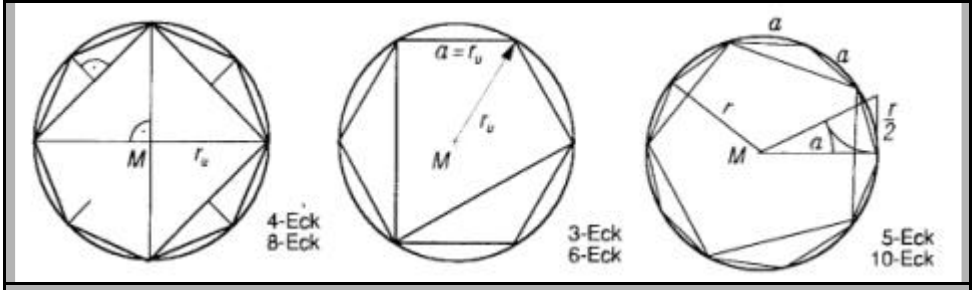
Voraussetzungen und Verbesserungen

Zu Beginn sei darauf hingewiesen, dass die für die Demoversion gewählte Lokation einen äußerst extremen Aufbau hat, um so vielen Problemstellungen wie möglich zu begegnen, um herauszufinden, wie ein Pathfindingsystem konzipiert werden muss. Normalerweise stellt sich in der Praxis wahrscheinlich nur ein Bruchteil der Probleme, weil die Lokations viel simpler gehalten sind. Eine solche Vorgehensweise ist dennoch sehr sinnvoll, weil es trotz aller Skalierbarkeit und Anpassbarkeit von Quellcode immer besser ist zu Beginn den Problemen zu begegnen, als im Nachhinein.

Die *forced portal A** Suche ermittelt anhand der Gegebenheiten immer den kürzesten, natürlichen Weg. Das Problem ist, dass, aufgrund von unterschiedlich großen Areas, manchmal andere Wege gewählt werden, als man vielleicht erwarten würde. Grund hierfür ist, dass viele kleine aneinandergereihte Areas, meist eine leichte Krümmung ergeben, sodass über diese Areas mittels der PTPP ein angeblich kürzerer Weg gefunden wird, als über große Areas, obwohl der Weg über die großen Areas vielleicht etwas kürzer ist. Aufgrund der Tatsache, dass eine PPTP durchgeführt und anhand der ermittelten Areas der Weg generiert wird, wird der Vorteil der anderen Strecke niemals bemerkt. Um diesem Problem zu begegnen, sollten gleichgroße Areas geschaffen werden. (Im Demoprogramm ist dies nicht der Fall, weil das Tool zur Erstellung der Areas dermaßen anfällig und schwer bedienbar ist, dass ein Wunsch nach gleichgroßen Areas kaum erfüllt werden kann.) Eine Empfehlung wären regelmäßige sechs-, acht- oder zehneckige Areas. (Um so mehr Portale vorhanden sind, umso genauer wird der Weg.)

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Abbildung 39:
Regelmäßige Vielecke



Normalerweise werden die Areas anhand von Level- oder Lokationeditoren erstellt. Aufgrund der Tatsache, dass es sehr schwierig ist gleichgroße, regelmäßige Vielecke zu erstellen, sollten vorgegebene Formen für die Areas in den Editoren verwendet werden. Eine bessere Lösung ist jedoch, dieses Problem algorithmisch zu lösen.

Eine weitere Verbesserung hinsichtlich noch exakterer Wege wäre, für die PPTP nicht nur die Mittelpunkte der Portale, sondern vielleicht zwei bis vier weitere Portalpunkte zu verwenden. Der Suchaufwand würde sich dadurch performanztechnisch nicht merklich verschlechtern.

Mitunter kommt es vor, dass sich die Heuristik dem Zwang einen Weg durch die Portale zu finden, wiedersetzt. Dies fällt auf, wenn eine andere Area berührt oder erreicht wird, als dies durch die PPTP vorgesehen ist. Momentan wird dies abgefangen, indem nur die Punkte der momentan untersuchten Area für die Suche zugelassen werden. Normalerweise müsste man, wenn dies bemerkt wird, sich den Weg bis zum momentanen Punkt merken, um dann eine neue PPTP mit dem momentanen Punkt als Startpunkt, der ermittelten Area als erste Zielarea und dem eigentlichen Zielpunkt erneut durchführen. Aufgrund der Tatsache, dass bei der Umsetzung der *forced portal A** Suche zu stark darauf geachtet wurde, den Speicher für die Suche so klein wie möglich zu halten, sodass sich alle Charaktere einen Speicher für die Gesamtsuche teilen, ist dies leider ohne einen riesigen Umbau nicht möglich. Das Problem ist, dass für jede Suche der Informationsspeicher von den

Knoten entkoppelt werden muss, damit keine Werte, die zu einem falschen Suchergebnis führen, vorhanden sind. Aus diesem Grund wird die Open List in der PQ und Open und Closed List in der Hash Table resetet, was bedeutet, dass alle Informationen verloren gehen. Die Open List wird sowieso bei jeder Suche in beiden Datentypen gelöscht, deshalb ist das eigentliche Problem die Closed List. Es wäre möglich, vor dem Löschen alle Informationen zu kopieren, was allerdings nicht effizient wäre. Weiterhin wäre es möglich, das Problem an sich zu ignorieren, weil die Ermittlung einer falschen Area relativ selten vorkommt. Prinzipiell müsste diesem Problem begegnet werden, indem die Closed List von der allgemeinen Hash Table getrennt wird. Prinzipiell könnte man die gesamte Suche erneut durchführen, indem die ermittelte bessere Area kurzfristig zu den bevorzugten Areas (siehe Motivationsmodell) hinzugefügt wird, mit dem Effekt, dass diese Area definitiv betreten wird. Das Problem ist, dass höchstwahrscheinlich, bis zu diesem Punkt, mit der PTP exakt der gleiche Weg ermittelt wird und sich die Frage stellt, ob eine Kopie des Weges nicht doch die bessere Alternative wäre. Aufgrund der Tatsache, dass regelmäßige Vielecke dieses Problem sowieso minimieren, stellt sich die Frage, ob man das Problem überhaupt anders angehen sollte, als es bei der *forced portal A** Suche gemacht wird, d.h. die Areas, die nicht mit der PPTP ermittelt wurden, zu ignorieren.

Wie schon darauf hingewiesen wurde, ist die Verwendung von regular grids zur Abbildung der Laufbereiche der Lokation faszinierend, weil sie eine Abfrage hinsichtlich der Begehbarkeit von O(1) garantieren. Allerdings muss eine Lösung gefunden werden, wie man mit den (Gleitkomma-)Bereichen zwischen den Pixel verfährt. Eine gute Methode ist sicherlich abzuwägen, zu welchem Pixel ein bestimmter Bereich gehört. Gibt es in der regular grid beispielsweise einen begehbaren Punkt $P_1(10, 20)$ und einen nicht begehbar Punkt $P_2(11, 20)$ und man möchte die Punkte $P_3(11.2, 20)$ und Punkte $P_4(11.58234, 20)$ auf die Begehbarkeit hin kontrollieren, dann werden P_3 und P_4 auf P_1 abgerundet. Eine besser Methode wäre herauszufinden, zu welchen Punkten P_3 und P_4 tendieren und somit abzuwägen, ob sie begehbar sind; P_3 entspräche dann P_1 und P_4 P_2. Man sollte nicht unbedingt mit dem

(In – einer- Area-) Enthaltensein- Algorithmus entscheiden, ob die Punkte tatsächlich in einer Area liegen, weil dies viel zu lange dauern würde.

Ein weiteres großes Problem tritt auf, wenn die Geraden zweier aufeinander folgender Portale beinahe im rechten Winkel zueinander stehen oder weit entfernt von einander sind, aber nahezu parallel verlaufen und sich vor dem ersten Portal ein Hindernis befindet. (Dies wird nur aufgeführt, weil die Problematik besteht, solange man keine regelmäßigen, vieleckigen Areas verwendet.) Die Folge ist, dass der erste Teil des erstellten Pfades äußerst unnatürlich aussieht, weil der Pfad ggf. sehr stark in Richtung des übernächsten Portals tendiert. Man könnte dieses Problem bereinigen, indem man den Penaltyfaktor für die Heuristik sehr groß werden lässt. Dies hätte allerdings zur Folge, das die Suche viel langsamer werden würde. Deshalb sollte man, wie schon erwähnt, regelmäßige Vielecke verwenden.

Es gibt eine Menge Ausnahmesituationen und eine Vielzahl von Möglichkeiten ein Pathfindingsystem zu optimieren. Dies ist wahrscheinlich der Grund, weshalb es erst nach Monaten nahezu perfekt funktioniert. Problematisch ist, dass man sich sehr stark in die Problematiken hineinsteigern kann, weil jede Optimierung, die zu einem besseren Ergebnis führt, wieder die Möglichkeit der Optimierung an einer anderen Stelle erlaubt, wobei die Einflussnahme auf die Heuristik wahrscheinlich am häufigsten durchgeführt wird. Die effektivste Optimierung, die durchzuführen ist, ist die Einführung von regelmäßigen Vielecken (als Areas), weil sie am meisten Vorteile garantiert. Des weiteren sollte eine Performanzoptimierung des Codes durchgeführt werden. Abgesehen von den Optimierungen gibt es kein System, welches dermaßen ästhetische Pfade generiert, wie das Basissystem der *forced portal A** Suche.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

8.0

Resümee

Die phänomenale Gestalt in Computerspielen

Die Grundfragen, die mit Hilfe dieser Diplomarbeit beantwortet wurden, sind:

1. Was ist die *phänomenale Gestalt* ?
2. Wie lässt sich die *phänomenale Gestalt* umsetzen ?

Der Begriff der *phänomenalen Gestalt* ist eine Beschreibung für einen Charakter in absoluter Vollendung. Eine solche Gestalt ist eine perfekte Abbildung menschlichen Lebens, die ein „künstliches“ Bewusstsein besitzt, welches sie in die Lage versetzt, ihr Verhalten und das Verhalten anderer zu reflektieren. Das Ziel der Spieleprogrammierung ist die Erschaffung einer *phänomenalen Gestalt*, die fähig ist, ihre Handlungsweise, hinsichtlich der perfekten Abbildung natürlichen Verhaltens, zu beeinflussen.

Prinzipiell sollte die AI Spieleprogrammierung versuchen eine Entität ganzheitlich im Sinne der *phänomenalen Gestalt* abzubilden. Die künstliche Intelligenz der Spieleprogrammierung beschäftig sich somit mit den Bereichen Lokomotion, Steering und Motivation. Die Diplomarbeit verfolgte primär den Bereich des Steerings. Wie muss der perfekte Weg von einem Punkt A nach B aussehen und wie kann die *phänomenale Gestalt* diesen Weg entwickeln? Da die Wissenschaften noch nicht so weit sind, zu benennen, warum ein menschliches Individuum einen bestimmten Weg wählt, sollte der perfekte kürzeste Weg einer Entität der sein, der entsprechend seiner Charakteristik erwartet wird. Ein kleiner Junge wählt im wahren

Leben höchstwahrscheinlich den für ihn interessantesten Weg. Kinder hassen es, wenn sie an der Hand ihrer Eltern sonntags im Park auf dem Weg bleiben müssen, damit sie ihre Kleidung nicht dreckig machen. Viel lieber toben sie herum, laufen über matschigen Untergrund, springen in Pfützen oder machen Wettweitrutschen auf Eisflächen mit ihren Freunden. Wenn Kinder in Computerspielen abgebildet werden, dann müssen sie auch in der virtuellen Welt lieber den Weg wählen, der Spaß macht. Avron verhält sich in diesem Sinne und nur so kann der Anspruch an eine AI – Spieleengine sein.

Aber warum muss der Weg der Kürzeste sein? Das primäre Ziel eines Spielers ist natürlich im Verlauf des Spiels Fortschritte zu machen und nicht Ewigkeiten darauf zu warten, dass der PC oder ein NPC endlich an seinem Ziel ankommt. Der Spieler darf durch das perfekte Verhalten des Charakters nicht in seinem Spielfluss gestört werden.

Eine weitere Frage ist, warum einer solchen perfekten Umsetzung menschlichen Verhaltens nachgegangen wird. Der Mensch flüchtet sich gerne in Fantasien, um seinem normalen Leben zu entfliehen und die Sorgen, Sorgen sein zu lassen. Er möchte ohne Mühen versuchen einen kleinen Moment jemand anderes zu sein. Der eine möchte vielleicht wie Michael Schumacher bei *Ridge Racer V*[80] die Weltmeisterschaft gewinnen, der andere wie *Harry Potter*[81] durch die Welt fliegen, ein dritter wie Uli Höhnes sein Team im *Fußball Manager 2002*[82] zum Erfolg bringen. Die Menschen verfolgen gerne einmal die Illusion, sich in einer erfolgreichen, spannenden, anderen Welt zu bewegen. Diese Illusion muss nicht nur aktiv gesteuert sein, sondern kann auch passiv in Kinofilmen erlebt werden. Je realistischer diese Illusionen abgebildet werden, um so einfacher und schöner ist es in der Regel für den Menschen, diese Illusion auch zu genießen. Die Diplomarbeit geht nicht auf die Frage ein, ob Computerspiele auch negative Auswirkungen haben können und ob der Realismus von Computerspielen Schuld daran ist, warum Kinder heutzutage zum Teil

[80] *Ridge Racer V:* Rennspiel der Firma Namco

[81] *Harry Potter*: Adventure der Firma Electronic Arts

[82] *Fußball Manager 2002: Fußballmanager der Firma* Electronic Arts

ihre Zeit lieber vor der Spielkonsole verbringen, als draußen. Sie beschäftigt sich nur damit, wie der Realismus so gut wie möglich umgesetzt werden kann. Wie gesagt, je realistischer die Spiele oder Kinofilme sind, um so beeindruckender sind sie für den Menschen. Der Spieler will sich in der virtuellen Welt aufgehoben wissen, somit müssen die Charaktere das gewisse Etwas haben, damit sie dem Spieler sympathisch sind oder unsympathisch. In der Regel will man sich oder andere wiedererkennen, entweder als einer der Helden oder in einer der Situationen der Helden. Nur mit Hilfe der perfekten Abbildung von Charakteren, nur unter Einsatz der *phänomenalen Gestalt* ist diese Illusion zu erreichen. Sobald ein Charakter seinen Pfad verlässt und beispielsweise durch Wände geht, ist die Illusion zerstört und der Spieler aus der Situation gerissen.

Im Bereich des Steerings besteht der Weg zur Erreichung dieses Ziels in der perfekten, charakterabhängigen Abbildung der Pfadsuche und gerade die Pfadsuche muss, weil die Illusion durch sie sehr schnell zerstört werden kann, perfekt sein. Aus diesem Grund bedarf die *phänomenale Gestalt* in erster Hinsicht einer sehr guten Pathfindingmethode, die *forced portal A** Suche, und den perfekten Algorithmus, um den kürzesten Weg zu bestimmen. Es wurde festgestellt, dass der A* dieser optimale Algorithmus ist, weil er im Gegensatz zur Tiefen- und Breitensuche, dem Dijkstra und der best first search Methode, in der Lage ist, den kürzesten Weg in kürzester Zeit zu berechnen. Der A*- Algorithmus hat zwei hervorragender Eigenschaften. Zum einen führt er eine sehr schnelle kürzeste Pfadsuche durch, weil er auf der Graphentheorie basiert. Zum anderen beschleunigt er die Suche nochmals, indem eine heuristische Abschätzung einer Restentfernung mit in die Suche einfließt, womit die Suche in Richtung Zielpunkt gelenkt wird. Im Gegensatz zu den anderen Suchalgorithmen werden somit viel weniger Knoten verglichen, was zu einer Beschleunigung der Suche führt, die sich positiv auf die Performanz auswirkt. Da die *phänomenale Gestalt* intelligent handelt, versucht sie, bevor sie mit der A* Suche beginnt, herauszufinden, ob es zwischen den Punkten überhaupt einen Weg geben kann und ob es nicht einen direkten Weg gibt, um die Suche noch schneller zu gestalten.

Der zweite Weg zur Implementierung eines perfekten Charakters hinsichtlich des Steerings, ist die Optimierung der A* Methode.

Die Speicher- und Performanzoptimierung führt man durch, indem man zum einen die Suchmenge einschränkt, um einen sehr schnellen Zugriff auf gültige Punkte zu haben und somit die Suche an sich zu beschleunigen. Die Speicher- und Performanzoptimierung ist deshalb so interessant und wichtig für die *phänomenale Gestalt*, weil mit ihr eine schnellere Suche durchgeführt werden kann, sodass der Eindruck entsteht, dass der Charakter eine Entscheidung für einen Weg sehr schnell trifft, was ihn wieder ein wenig menschlicher erscheinen lässt. Nachdem die Suchmenge eingegrenzt wurde, muss man die Datentypen des A* hinsichtlich der Geschwindigkeit verbessern. Dies wird umgesetzt, indem eine Hash Table implementiert (die sich dem Speicher einer Knotenbank bedient) wird, die sowohl die Knoten der Open als auch der Closed List verwaltet. Die Open List wird noch in einer weiteren Liste geführt. Der Datentyp für diese Liste ist eine priority queue. Somit hat man die Möglichkeit, abhängig vom Verwendungszweck, den schnelleren Zugriff auf die Informationen zu wählen. Die Speicher- und Performanzoptimierung ist nicht direkt ersichtlich, sie ist für den Spieler eher spürbar, indem der Charakter schneller auf die Anweisungen des Spielers oder auf Situationen im Spielverlauf reagiert.

Das Pathfinding ist deshalb so gut umgesetzt, weil die entwickelte *forced portal A** Suche einen sehr realistischen Weg generiert, bei dem man sich vorstellen kann, ihn selber zu gehen. Mit Hilfe dieser Methode ist es möglich, Einfluss auf die Kraft der Heuristik zu nehmen, indem die Wegführung bestraft wird, die unrealistisch erscheint. Ein Vorteil des *forced portal A** Konzeptes ist, dass eine weitere ästhetische Optimierung mit *straight* und *smooth paths* nicht notwendig ist. Des weiteren ist eine Motivation, die die Performanz beinahe überhaupt nicht beeinflusst, sehr leicht umsetzbar.

Diese Diplomarbeit hat bewiesen, dass es im Bereich des Steerings möglich ist, eine *phänomenale Gestalt* zu implementieren. Es ist gelungen aufzuzeigen, was die *phänomenale Gestalt* in Computerspielen sein sollte und wie dieses Ziel erreichbar ist.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Leider ist es noch immer so, dass bei vielen Computerspielen gerade das Pathfinding schlecht implementiert wird. Das Beispielprogramm gab einen kleinen Einblick, wie es aussehen kann. Hier reagieren drei Spielfiguren entsprechend ihres Charakters verschieden auf die gleiche Welt.

„Das ideale Computerspiel lässt einen erst handfeste Abenteuer erleben, doch am Ende vermittelt es einem eine Ahnung vom Übernatürlichen.“[83]

Douglas Coupland

[83] Zitat: Douglas Coupland, Microslaves, S.178

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

9.0

Literaturverzeichnis

Internet

Dorn, J.; Gottlob G., o. Datum. „Künstliche Intelligenz“
<http://www.dbai.tuwien.ac.at/staff/dorn/Publications/Papers/hi00.pdf> (09.11.2001)

Minsky, Marvin, Juli 1992. „Future of AI Technologie“
<http://www.media.mit.edu/~minsky/papers/CausalDiversity.html> (09.11.2001)

Musselwhite, Charles, Mai 1996. „Minsky's Critique of the Perceptron“
<http://cogprints.ecs.soton.ac.uk/~harnad/Hypermail/Explaining.Mind96/0140.html>

Stout, Bryan, August 1997. „smart moves – intelligent pathfinding“
<http://www.gamasutra.com/features/19970801/pathfinding.htm> (01.12.2001)

Woodcock , Steve M., o. Datum, „Game AI Resources: Robotics“
<http://gameai.com/robotics.html> (22.12.2001)

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Bücher

Breymann, Ulrich: C++ Eine Einführung, 4. Aufl., München; Wien: Carl Hanser Verlag, 1997.

Brockhaus: Der große Brockhaus, Band 16, 18. Auflage, Wiesbaden: F.A. Brockhaus, 1983.

Bucher, Theodor: Einführung in die angewandte Logik, 1. Aufl., Berlin; New York: de Gruyter, 1987.

Coupland, Douglas: Microsklaven, Taschenbuchausgabe 10/99, Goldmann Verlag, 1999.

Drosdowski, Günther; Müller, Wolfgang; Scholze- Stubenrecht, Werner; u.a.: Duden Das Fremdwörterbuch, Band 5, Mannheim; Leipzig; Wien; Zürich: Dudenverlag.

Eberly, David H.: 3D Game Engine Design, 1. Aufl., Morgan Kaufmann Publishers, 2001.

Ferbes, Jacque: Multiagentensysteme, 1. Aufl., Bonn; München; Paris: Addison Wesley GmbH, 2001.

Perez, Adrian; Royer, Dan: 3-D Spiele Programmierung mit DirectX, Frankfurt: Software & Support Verlag GmbH, 2000.

Rabin, Steve; Stout, Bryan, u.a.: Game Programming Gems, Rockland, Massachusetts: Charles River Media Inc., 2000.

Sedgewick, Robert: Algorithmen in C++, 5. Aufl., Bonn; München; Paris: Addison Wesley GmbH, 1999.

Schwarze, Jochen: Mathematik für Wirtschaftswissenschaftler, Band 3, 10. Aufl., Herne; Berlin: Verlag Neue Wirtschafts- Briefe GmbH, 1996.

Walter, Rolf: Einführung in die lineare Algebra, 3. Aufl., Braunschweig; Wiesbaden: Vieweg, 1990.

Zeitschriften

Evans, Richard: Future of AI in Games: A Personal View: Game Developer, 8 (2001), 8, S. 46- 49

Ginsberg, Matthew L: Computer, Spiele und die Realität: Spektrum der Wissenschaft Spezial, (2000), 1, S. 84- 89

Woodcock, Steven: GameAI The State of the Industry: Game Developer, 7 (2000), 8, S. 34- 39

Woodcock, Steven: GameAI: The State of the Industry 2000-2001 It's not Just Art, It's Engineering: Game Developer, 8 (2001), 8, S. 36- 44

Dokumentationen

Microsoft Visual C++ 6.0 Hilfe/ Dokumentation

Aufsätze

Metzinger, Thomas: Postbiotisches Bewußtsein: Wie man ein künstliches Subjekt baut – und warum wir es nicht tun sollten: Heinz Nixdorf MuseumsForum: Computer.Gehirn: Was kann der Mensch? Was können die Computer?, Paderborn; München; Wien; Zürich: Verlag Ferdinand Schöningh, 2001, S. 87- 113.

Ritter, Helge: Die Evolution der künstlichen Intelligenz: Heinz Nixdorf MuseumsForum: Computer.Gehirn: Was kann der Mensch? Was können die Computer?, Paderborn; München; Wien; Zürich: Verlag Ferdinand Schöningh, 2001, S. 39- 61.

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

10.0

Anhang

Debug- Ausgabe

Gesamte Debug- Ausgabe der vorgestellten Suche ohne Kommentare:

new search Nr.: 19
Start: X:216.00000000 Y:569.00000000
Goal: X:233.00000000 Y:347.00000000

search:
Start: X:216.00000000 Y:569.00000000
Goal: X:232.00000000 Y:347.00000000

preparing plane portal search

plane portal search add start node

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

starting plane portal search

Astar start: GoalX:232.00000000 GoalY:347.00000000

bestnode: X:216.00000000 Y:569.00000000 Total:49540.00000000
gkCActualNode: X:122.00000000 Y:545.50000000 Total:60890.50000000
gkCActualNode: X:319.00000000 Y:570.50000000 Total:68132.50000000

bestnode: X:122.00000000 Y:545.50000000 Total:60890.50000000
gkCActualNode: X:144.50000000 Y:464.50000000 Total:37918.00000000

bestnode: X:144.50000000 Y:464.50000000 Total:37918.00000000
gkCActualNode: X:156.00000000 Y:382.00000000 Total:30395.00000000
gkCActualNode: X:69.00000000 Y:389.50000000 Total:56156.00000000

bestnode: X:156.00000000 Y:382.00000000 Total:30395.00000000
gkCActualNode: X:200.00000000 Y:356.50000000 Total:27094.50000000

bestnode: X:200.00000000 Y:356.50000000 Total:27094.50000000

CreatePortalPath start

Count nodes: 1

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Count nodes: 2

Count nodes: 3

Count nodes: 4

Count nodes: 5

nArea:6 X:200.00000000 Y:356.50000000 Total:0.00000000

nArea:21 X:156.00000000 Y:382.00000000 Total:0.00000000

nArea:25 X:144.50000000 Y:464.50000000 Total:0.00000000

nArea:26 X:122.00000000 Y:545.50000000 Total:0.00000000

CreatePortalPath done

plane portal search done

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Portal search Nr.:1

Astar start: GoalX:144.50000000 GoalY:464.50000000

bestnode: X:216.00000000 Y:569.00000000 Total:27585.60000000
gkCActualNode: X:218.00000000 Y:567.00000000 Total:26734.28000000
gkCActualNode: X:214.00000000 Y:567.00000000 Total:25773.32000000
gkCActualNode: X:218.00000000 Y:571.00000000 Total:28138.76000000
gkCActualNode: X:214.00000000 Y:571.00000000 Total:27177.80000000
gkCActualNode: X:218.00000000 Y:569.00000000 Total:27425.80000000
gkCActualNode: X:214.00000000 Y:569.00000000 Total:26464.84000000
gkCActualNode: X:216.00000000 Y:571.00000000 Total:27647.56000000
gkCActualNode: X:216.00000000 Y:567.00000000 Total:26243.08000000

bestnode: X:214.00000000 Y:567.00000000 Total:25773.32000000
gkCActualNode: X:216.00000000 Y:565.00000000 Total:25573.00000000
gkCActualNode: X:212.00000000 Y:565.00000000 Total:24638.92000000
gkCActualNode: X:212.00000000 Y:569.00000000 Total:26016.52000000
gkCActualNode: X:212.00000000 Y:567.00000000 Total:25317.00000000
gkCActualNode: X:214.00000000 Y:565.00000000 Total:25095.24000000

bestnode: X:212.00000000 Y:565.00000000 Total:24638.92000000
gkCActualNode: X:214.00000000 Y:563.00000000 Total:24438.60000000
gkCActualNode: X:210.00000000 Y:563.00000000 Total:23531.40000000
gkCActualNode: X:210.00000000 Y:567.00000000 Total:24882.12000000
gkCActualNode: X:210.00000000 Y:565.00000000 Total:24196.04000000
gkCActualNode: X:212.00000000 Y:563.00000000 Total:23974.28000000

bestnode: X:210.00000000 Y:563.00000000 Total:23531.40000000
gkCActualNode: X:212.00000000 Y:561.00000000 Total:23331.08000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:208.00000000 Y:561.00000000 Total:22450.76000000
gkCActualNode: X:208.00000000 Y:565.00000000 Total:23774.60000000
gkCActualNode: X:208.00000000 Y:563.00000000 Total:23101.96000000
gkCActualNode: X:210.00000000 Y:561.00000000 Total:22880.20000000

bestnode: X:208.00000000 Y:561.00000000 Total:22450.76000000
gkCActualNode: X:210.00000000 Y:559.00000000 Total:22250.44000000
gkCActualNode: X:206.00000000 Y:559.00000000 Total:21397.00000000
gkCActualNode: X:206.00000000 Y:563.00000000 Total:22693.96000000
gkCActualNode: X:206.00000000 Y:561.00000000 Total:22034.76000000
gkCActualNode: X:208.00000000 Y:559.00000000 Total:21813.00000000

bestnode: X:206.00000000 Y:559.00000000 Total:21397.00000000
gkCActualNode: X:208.00000000 Y:557.00000000 Total:21196.68000000
gkCActualNode: X:204.00000000 Y:557.00000000 Total:20370.12000000
gkCActualNode: X:204.00000000 Y:561.00000000 Total:21640.20000000
gkCActualNode: X:204.00000000 Y:559.00000000 Total:20994.44000000
gkCActualNode: X:206.00000000 Y:557.00000000 Total:20772.68000000

bestnode: X:204.00000000 Y:557.00000000 Total:20370.12000000
gkCActualNode: X:202.00000000 Y:559.00000000 Total:20613.32000000
gkCActualNode: X:202.00000000 Y:557.00000000 Total:19981.00000000

bestnode: X:202.00000000 Y:557.00000000 Total:19981.00000000
gkCActualNode: X:200.00000000 Y:559.00000000 Total:20237.64000000
gkCActualNode: X:200.00000000 Y:557.00000000 Total:19605.32000000

bestnode: X:200.00000000 Y:557.00000000 Total:19605.32000000
gkCActualNode: X:198.00000000 Y:559.00000000 Total:19875.40000000
gkCActualNode: X:198.00000000 Y:557.00000000 Total:19243.08000000

bestnode: X:198.00000000 Y:557.00000000 Total:19243.08000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:196.00000000 Y:559.00000000 Total:19526.60000000
gkCActualNode: X:196.00000000 Y:557.00000000 Total:18894.28000000

bestnode: X:196.00000000 Y:557.00000000 Total:18894.28000000
gkCActualNode: X:194.00000000 Y:559.00000000 Total:19191.24000000
gkCActualNode: X:194.00000000 Y:557.00000000 Total:18558.92000000

bestnode: X:194.00000000 Y:557.00000000 Total:18558.92000000
gkCActualNode: X:192.00000000 Y:559.00000000 Total:18869.32000000
gkCActualNode: X:192.00000000 Y:557.00000000 Total:18237.00000000

bestnode: X:192.00000000 Y:557.00000000 Total:18237.00000000
gkCActualNode: X:190.00000000 Y:559.00000000 Total:18560.84000000
gkCActualNode: X:190.00000000 Y:557.00000000 Total:17928.52000000

bestnode: X:190.00000000 Y:557.00000000 Total:17928.52000000
gkCActualNode: X:188.00000000 Y:559.00000000 Total:18265.80000000
gkCActualNode: X:188.00000000 Y:557.00000000 Total:17633.48000000

bestnode: X:188.00000000 Y:557.00000000 Total:17633.48000000
gkCActualNode: X:186.00000000 Y:559.00000000 Total:17984.20000000
gkCActualNode: X:186.00000000 Y:557.00000000 Total:17351.88000000

bestnode: X:186.00000000 Y:557.00000000 Total:17351.88000000
gkCActualNode: X:184.00000000 Y:559.00000000 Total:17716.04000000
gkCActualNode: X:184.00000000 Y:557.00000000 Total:17083.72000000

bestnode: X:184.00000000 Y:557.00000000 Total:17083.72000000
gkCActualNode: X:182.00000000 Y:559.00000000 Total:17461.32000000
gkCActualNode: X:182.00000000 Y:557.00000000 Total:16829.00000000

bestnode: X:182.00000000 Y:557.00000000 Total:16829.00000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:180.00000000 Y:559.00000000 Total:17220.04000000
gkCActualNode: X:180.00000000 Y:557.00000000 Total:16587.72000000

bestnode: X:180.00000000 Y:557.00000000 Total:16587.72000000
gkCActualNode: X:178.00000000 Y:559.00000000 Total:16992.20000000
gkCActualNode: X:178.00000000 Y:557.00000000 Total:16359.88000000

bestnode: X:178.00000000 Y:557.00000000 Total:16359.88000000
gkCActualNode: X:176.00000000 Y:559.00000000 Total:16777.80000000
gkCActualNode: X:176.00000000 Y:557.00000000 Total:16145.48000000

bestnode: X:176.00000000 Y:557.00000000 Total:16145.48000000
gkCActualNode: X:174.00000000 Y:559.00000000 Total:16576.84000000
gkCActualNode: X:174.00000000 Y:557.00000000 Total:15944.52000000

bestnode: X:174.00000000 Y:557.00000000 Total:15944.52000000
gkCActualNode: X:172.00000000 Y:559.00000000 Total:16389.32000000
gkCActualNode: X:172.00000000 Y:557.00000000 Total:15757.00000000

bestnode: X:172.00000000 Y:557.00000000 Total:15757.00000000
gkCActualNode: X:170.00000000 Y:559.00000000 Total:16215.24000000
gkCActualNode: X:170.00000000 Y:557.00000000 Total:15582.92000000

bestnode: X:170.00000000 Y:557.00000000 Total:15582.92000000
gkCActualNode: X:168.00000000 Y:559.00000000 Total:16054.60000000
gkCActualNode: X:168.00000000 Y:557.00000000 Total:15422.28000000

bestnode: X:168.00000000 Y:557.00000000 Total:15422.28000000
gkCActualNode: X:166.00000000 Y:559.00000000 Total:15907.40000000
gkCActualNode: X:166.00000000 Y:557.00000000 Total:15275.08000000

bestnode: X:166.00000000 Y:557.00000000 Total:15275.08000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:164.00000000 Y:559.00000000 Total:15773.64000000
gkCActualNode: X:164.00000000 Y:557.00000000 Total:15141.32000000

bestnode: X:164.00000000 Y:557.00000000 Total:15141.32000000
gkCActualNode: X:162.00000000 Y:555.00000000 Total:14410.12000000
gkCActualNode: X:162.00000000 Y:559.00000000 Total:15653.32000000
gkCActualNode: X:162.00000000 Y:557.00000000 Total:15021.00000000

bestnode: X:162.00000000 Y:555.00000000 Total:14410.12000000
gkCActualNode: X:160.00000000 Y:557.00000000 Total:14922.12000000
gkCActualNode: X:160.00000000 Y:555.00000000 Total:14303.24000000

bestnode: X:160.00000000 Y:555.00000000 Total:14303.24000000
gkCActualNode: X:158.00000000 Y:553.00000000 Total:13612.36000000
gkCActualNode: X:158.00000000 Y:557.00000000 Total:14828.68000000
gkCActualNode: X:158.00000000 Y:555.00000000 Total:14209.80000000

bestnode: X:158.00000000 Y:553.00000000 Total:13612.36000000
gkCActualNode: X:156.00000000 Y:555.00000000 Total:14137.80000000
gkCActualNode: X:156.00000000 Y:553.00000000 Total:13532.36000000

bestnode: X:156.00000000 Y:553.00000000 Total:13532.36000000
gkCActualNode: X:154.00000000 Y:555.00000000 Total:14071.24000000
gkCActualNode: X:154.00000000 Y:553.00000000 Total:13465.80000000

bestnode: X:154.00000000 Y:553.00000000 Total:13465.80000000
gkCActualNode: X:152.00000000 Y:551.00000000 Total:12828.68000000
gkCActualNode: X:152.00000000 Y:555.00000000 Total:14018.12000000
gkCActualNode: X:152.00000000 Y:553.00000000 Total:13412.68000000

bestnode: X:152.00000000 Y:551.00000000 Total:12828.68000000
gkCActualNode: X:150.00000000 Y:553.00000000 Total:13381.00000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:150.00000000 Y:551.00000000 Total:12789.00000000

bestnode: X:150.00000000 Y:551.00000000 Total:12789.00000000
gkCActualNode: X:148.00000000 Y:553.00000000 Total:13354.76000000
gkCActualNode: X:148.00000000 Y:551.00000000 Total:12762.76000000

bestnode: X:148.00000000 Y:551.00000000 Total:12762.76000000
gkCActualNode: X:146.00000000 Y:553.00000000 Total:13341.96000000
gkCActualNode: X:146.00000000 Y:551.00000000 Total:12749.96000000

bestnode: X:146.00000000 Y:551.00000000 Total:12749.96000000
gkCActualNode: X:144.00000000 Y:549.00000000 Total:12180.04000000
gkCActualNode: X:144.00000000 Y:553.00000000 Total:13342.60000000
gkCActualNode: X:144.00000000 Y:551.00000000 Total:12750.60000000

bestnode: X:144.00000000 Y:549.00000000 Total:12180.04000000
gkCActualNode: X:142.00000000 Y:551.00000000 Total:12772.68000000
gkCActualNode: X:142.00000000 Y:549.00000000 Total:12194.12000000

bestnode: X:142.00000000 Y:549.00000000 Total:12194.12000000
gkCActualNode: X:140.00000000 Y:547.00000000 Total:7022.50000000
gkCActualNode: X:144.00000000 Y:551.00000000 Total:7678.50000000
gkCActualNode: X:140.00000000 Y:551.00000000 Total:12800.20000000
gkCActualNode: X:140.00000000 Y:549.00000000 Total:12221.64000000

bestnode: X:140.00000000 Y:547.00000000 Total:7022.50000000

Portal search Nr.:2

direct path:

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Portal search Nr.:3

Astar start: GoalX:200.00000000 GoalY:356.50000000

bestnode: X:148.00000000 Y:463.00000000 Total:24138.24000000
gkCActualNode: X:150.00000000 Y:461.00000000 Total:45485.52000000
gkCActualNode: X:146.00000000 Y:461.00000000 Total:46184.40000000
gkCActualNode: X:150.00000000 Y:465.00000000 Total:46916.88000000
gkCActualNode: X:146.00000000 Y:465.00000000 Total:47615.76000000
gkCActualNode: X:150.00000000 Y:463.00000000 Total:46190.48000000
gkCActualNode: X:146.00000000 Y:463.00000000 Total:46889.36000000
gkCActualNode: X:148.00000000 Y:465.00000000 Total:37411.75000000
gkCActualNode: X:148.00000000 Y:461.00000000 Total:36559.75000000

bestnode: X:148.00000000 Y:461.00000000 Total:36559.75000000
gkCActualNode: X:150.00000000 Y:459.00000000 Total:44794.00000000
gkCActualNode: X:146.00000000 Y:459.00000000 Total:45492.88000000
gkCActualNode: X:148.00000000 Y:459.00000000 Total:36149.75000000

bestnode: X:148.00000000 Y:459.00000000 Total:36149.75000000
gkCActualNode: X:150.00000000 Y:457.00000000 Total:44115.92000000
gkCActualNode: X:146.00000000 Y:457.00000000 Total:44814.80000000
gkCActualNode: X:148.00000000 Y:457.00000000 Total:35747.75000000

bestnode: X:148.00000000 Y:457.00000000 Total:35747.75000000
gkCActualNode: X:150.00000000 Y:455.00000000 Total:43451.28000000
gkCActualNode: X:146.00000000 Y:455.00000000 Total:44150.16000000
gkCActualNode: X:148.00000000 Y:455.00000000 Total:35353.75000000

bestnode: X:148.00000000 Y:455.00000000 Total:35353.75000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:150.00000000 Y:453.00000000 Total:42800.08000000
gkCActualNode: X:146.00000000 Y:453.00000000 Total:43498.96000000
gkCActualNode: X:148.00000000 Y:453.00000000 Total:34967.75000000

bestnode: X:148.00000000 Y:453.00000000 Total:34967.75000000
gkCActualNode: X:150.00000000 Y:451.00000000 Total:42162.32000000
gkCActualNode: X:146.00000000 Y:451.00000000 Total:42861.20000000
gkCActualNode: X:148.00000000 Y:451.00000000 Total:34589.75000000

bestnode: X:148.00000000 Y:451.00000000 Total:34589.75000000
gkCActualNode: X:150.00000000 Y:449.00000000 Total:41538.00000000
gkCActualNode: X:146.00000000 Y:449.00000000 Total:42236.88000000
gkCActualNode: X:148.00000000 Y:449.00000000 Total:34219.75000000

bestnode: X:148.00000000 Y:449.00000000 Total:34219.75000000
gkCActualNode: X:150.00000000 Y:447.00000000 Total:40927.12000000
gkCActualNode: X:146.00000000 Y:447.00000000 Total:41626.00000000
gkCActualNode: X:148.00000000 Y:447.00000000 Total:33857.75000000

bestnode: X:148.00000000 Y:447.00000000 Total:33857.75000000
gkCActualNode: X:150.00000000 Y:445.00000000 Total:40329.68000000
gkCActualNode: X:146.00000000 Y:445.00000000 Total:41028.56000000
gkCActualNode: X:148.00000000 Y:445.00000000 Total:33503.75000000

bestnode: X:148.00000000 Y:445.00000000 Total:33503.75000000
gkCActualNode: X:150.00000000 Y:443.00000000 Total:39745.68000000
gkCActualNode: X:146.00000000 Y:443.00000000 Total:40444.56000000
gkCActualNode: X:148.00000000 Y:443.00000000 Total:33157.75000000

bestnode: X:148.00000000 Y:443.00000000 Total:33157.75000000
gkCActualNode: X:150.00000000 Y:441.00000000 Total:39175.12000000
gkCActualNode: X:146.00000000 Y:441.00000000 Total:39874.00000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:148.00000000 Y:441.00000000 Total:32819.75000000

bestnode: X:148.00000000 Y:441.00000000 Total:32819.75000000
gkCActualNode: X:150.00000000 Y:439.00000000 Total:38618.00000000
gkCActualNode: X:146.00000000 Y:439.00000000 Total:39316.88000000
gkCActualNode: X:148.00000000 Y:439.00000000 Total:32489.75000000

bestnode: X:148.00000000 Y:439.00000000 Total:32489.75000000
gkCActualNode: X:150.00000000 Y:437.00000000 Total:38074.32000000
gkCActualNode: X:146.00000000 Y:437.00000000 Total:38773.20000000
gkCActualNode: X:148.00000000 Y:437.00000000 Total:32167.75000000

bestnode: X:148.00000000 Y:437.00000000 Total:32167.75000000
gkCActualNode: X:150.00000000 Y:435.00000000 Total:37544.08000000
gkCActualNode: X:146.00000000 Y:435.00000000 Total:38242.96000000
gkCActualNode: X:148.00000000 Y:435.00000000 Total:31853.75000000

bestnode: X:148.00000000 Y:435.00000000 Total:31853.75000000
gkCActualNode: X:150.00000000 Y:433.00000000 Total:37027.28000000
gkCActualNode: X:146.00000000 Y:433.00000000 Total:37726.16000000
gkCActualNode: X:148.00000000 Y:433.00000000 Total:31547.75000000

bestnode: X:148.00000000 Y:433.00000000 Total:31547.75000000
gkCActualNode: X:150.00000000 Y:431.00000000 Total:36523.92000000
gkCActualNode: X:146.00000000 Y:431.00000000 Total:37222.80000000
gkCActualNode: X:148.00000000 Y:431.00000000 Total:31249.75000000

bestnode: X:148.00000000 Y:431.00000000 Total:31249.75000000
gkCActualNode: X:150.00000000 Y:429.00000000 Total:36034.00000000
gkCActualNode: X:146.00000000 Y:429.00000000 Total:36732.88000000
gkCActualNode: X:148.00000000 Y:429.00000000 Total:30959.75000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

bestnode: X:148.00000000 Y:429.00000000 Total:30959.75000000
gkCActualNode: X:150.00000000 Y:427.00000000 Total:35557.52000000
gkCActualNode: X:146.00000000 Y:427.00000000 Total:36256.40000000
gkCActualNode: X:148.00000000 Y:427.00000000 Total:30677.75000000

bestnode: X:148.00000000 Y:427.00000000 Total:30677.75000000
gkCActualNode: X:150.00000000 Y:425.00000000 Total:35094.48000000
gkCActualNode: X:146.00000000 Y:425.00000000 Total:35793.36000000
gkCActualNode: X:148.00000000 Y:425.00000000 Total:30403.75000000

bestnode: X:148.00000000 Y:425.00000000 Total:30403.75000000
gkCActualNode: X:150.00000000 Y:423.00000000 Total:34644.88000000
gkCActualNode: X:146.00000000 Y:423.00000000 Total:35343.76000000
gkCActualNode: X:148.00000000 Y:423.00000000 Total:30137.75000000

bestnode: X:148.00000000 Y:423.00000000 Total:30137.75000000
gkCActualNode: X:150.00000000 Y:421.00000000 Total:34208.72000000
gkCActualNode: X:146.00000000 Y:421.00000000 Total:34907.60000000
gkCActualNode: X:148.00000000 Y:421.00000000 Total:29879.75000000

bestnode: X:148.00000000 Y:421.00000000 Total:29879.75000000
gkCActualNode: X:150.00000000 Y:419.00000000 Total:33786.00000000
gkCActualNode: X:146.00000000 Y:419.00000000 Total:34484.88000000
gkCActualNode: X:148.00000000 Y:419.00000000 Total:29629.75000000

bestnode: X:148.00000000 Y:419.00000000 Total:29629.75000000
gkCActualNode: X:150.00000000 Y:417.00000000 Total:33376.72000000
gkCActualNode: X:146.00000000 Y:417.00000000 Total:34075.60000000
gkCActualNode: X:148.00000000 Y:417.00000000 Total:29387.75000000

bestnode: X:148.00000000 Y:417.00000000 Total:29387.75000000
gkCActualNode: X:150.00000000 Y:415.00000000 Total:32980.88000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:146.00000000 Y:415.00000000 Total:33679.76000000
gkCActualNode: X:148.00000000 Y:415.00000000 Total:29153.75000000

bestnode: X:148.00000000 Y:415.00000000 Total:29153.75000000
gkCActualNode: X:150.00000000 Y:413.00000000 Total:32598.48000000
gkCActualNode: X:146.00000000 Y:413.00000000 Total:33297.36000000
gkCActualNode: X:148.00000000 Y:413.00000000 Total:28927.75000000

bestnode: X:148.00000000 Y:413.00000000 Total:28927.75000000
gkCActualNode: X:150.00000000 Y:411.00000000 Total:28509.75000000
gkCActualNode: X:146.00000000 Y:411.00000000 Total:32928.40000000
gkCActualNode: X:146.00000000 Y:415.00000000 Total:29377.75000000
gkCActualNode: X:148.00000000 Y:411.00000000 Total:28709.75000000

bestnode: X:150.00000000 Y:411.00000000 Total:28509.75000000
gkCActualNode: X:152.00000000 Y:409.00000000 Total:28107.75000000
gkCActualNode: X:148.00000000 Y:409.00000000 Total:32220.72000000
gkCActualNode: X:152.00000000 Y:413.00000000 Total:32281.20000000
gkCActualNode: X:152.00000000 Y:411.00000000 Total:31904.24000000
gkCActualNode: X:150.00000000 Y:413.00000000 Total:28735.75000000
gkCActualNode: X:150.00000000 Y:409.00000000 Total:28299.75000000

bestnode: X:152.00000000 Y:409.00000000 Total:28107.75000000
gkCActualNode: X:154.00000000 Y:407.00000000 Total:27721.75000000
gkCActualNode: X:150.00000000 Y:407.00000000 Total:31539.92000000
gkCActualNode: X:154.00000000 Y:411.00000000 Total:31600.40000000
gkCActualNode: X:154.00000000 Y:409.00000000 Total:31236.88000000
gkCActualNode: X:152.00000000 Y:411.00000000 Total:28325.75000000
gkCActualNode: X:152.00000000 Y:407.00000000 Total:27905.75000000

bestnode: X:154.00000000 Y:407.00000000 Total:27721.75000000
gkCActualNode: X:156.00000000 Y:405.00000000 Total:27351.75000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:152.00000000 Y:405.00000000 Total:30886.00000000
gkCActualNode: X:156.00000000 Y:409.00000000 Total:30946.48000000
gkCActualNode: X:156.00000000 Y:407.00000000 Total:30596.40000000
gkCActualNode: X:154.00000000 Y:409.00000000 Total:27931.75000000
gkCActualNode: X:154.00000000 Y:405.00000000 Total:27527.75000000

bestnode: X:156.00000000 Y:405.00000000 Total:27351.75000000
gkCActualNode: X:158.00000000 Y:403.00000000 Total:26997.75000000
gkCActualNode: X:154.00000000 Y:403.00000000 Total:30258.96000000
gkCActualNode: X:158.00000000 Y:407.00000000 Total:30319.44000000
gkCActualNode: X:158.00000000 Y:405.00000000 Total:29982.80000000
gkCActualNode: X:156.00000000 Y:407.00000000 Total:27553.75000000
gkCActualNode: X:156.00000000 Y:403.00000000 Total:27165.75000000

bestnode: X:158.00000000 Y:403.00000000 Total:26997.75000000
gkCActualNode: X:160.00000000 Y:401.00000000 Total:26659.75000000
gkCActualNode: X:156.00000000 Y:401.00000000 Total:29658.80000000
gkCActualNode: X:160.00000000 Y:405.00000000 Total:29719.28000000
gkCActualNode: X:160.00000000 Y:403.00000000 Total:29396.08000000
gkCActualNode: X:158.00000000 Y:405.00000000 Total:27191.75000000
gkCActualNode: X:158.00000000 Y:401.00000000 Total:26819.75000000

bestnode: X:160.00000000 Y:401.00000000 Total:26659.75000000
gkCActualNode: X:162.00000000 Y:399.00000000 Total:26337.75000000
gkCActualNode: X:158.00000000 Y:399.00000000 Total:29085.52000000
gkCActualNode: X:162.00000000 Y:403.00000000 Total:29146.00000000
gkCActualNode: X:162.00000000 Y:401.00000000 Total:28836.24000000
gkCActualNode: X:160.00000000 Y:403.00000000 Total:26845.75000000
gkCActualNode: X:160.00000000 Y:399.00000000 Total:26489.75000000

bestnode: X:162.00000000 Y:399.00000000 Total:26337.75000000
gkCActualNode: X:164.00000000 Y:397.00000000 Total:26031.75000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:160.00000000 Y:397.00000000 Total:28539.12000000
gkCActualNode: X:164.00000000 Y:401.00000000 Total:28599.60000000
gkCActualNode: X:164.00000000 Y:399.00000000 Total:28303.28000000
gkCActualNode: X:162.00000000 Y:401.00000000 Total:26515.75000000
gkCActualNode: X:162.00000000 Y:397.00000000 Total:26175.75000000

bestnode: X:164.00000000 Y:397.00000000 Total:26031.75000000
gkCActualNode: X:166.00000000 Y:395.00000000 Total:25741.75000000
gkCActualNode: X:162.00000000 Y:395.00000000 Total:28019.60000000
gkCActualNode: X:166.00000000 Y:399.00000000 Total:28080.08000000
gkCActualNode: X:166.00000000 Y:397.00000000 Total:27797.20000000
gkCActualNode: X:164.00000000 Y:399.00000000 Total:26201.75000000
gkCActualNode: X:164.00000000 Y:395.00000000 Total:25877.75000000

bestnode: X:166.00000000 Y:395.00000000 Total:25741.75000000

Final search ONE PORTAL AWAY FROM goal area

Astar start: GoalX:216.00000000 GoalY:351.00000000

bestnode: X:166.00000000 Y:395.00000000 Total:4436.00000000
gkCActualNode: X:164.00000000 Y:393.00000000 Total:30617.74000000
gkCActualNode: X:164.00000000 Y:395.00000000 Total:30902.70000000
gkCActualNode: X:166.00000000 Y:397.00000000 Total:30862.38000000
gkCActualNode: X:166.00000000 Y:393.00000000 Total:30271.02000000

bestnode: X:166.00000000 Y:393.00000000 Total:30271.02000000
gkCActualNode: X:168.00000000 Y:391.00000000 Total:27019.50000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:164.00000000 Y:391.00000000 Total:30346.22000000
gkCActualNode: X:164.00000000 Y:395.00000000 Total:27755.50000000
gkCActualNode: X:166.00000000 Y:391.00000000 Total:29999.50000000

bestnode: X:168.00000000 Y:391.00000000 Total:27019.50000000
gkCActualNode: X:166.00000000 Y:389.00000000 Total:29749.42000000
gkCActualNode: X:168.00000000 Y:389.00000000 Total:29416.14000000

bestnode: X:164.00000000 Y:395.00000000 Total:27755.50000000
gkCActualNode: X:162.00000000 Y:393.00000000 Total:30985.90000000
gkCActualNode: X:162.00000000 Y:395.00000000 Total:31270.86000000

bestnode: X:168.00000000 Y:389.00000000 Total:29416.14000000
gkCActualNode: X:170.00000000 Y:387.00000000 Total:26539.50000000
gkCActualNode: X:166.00000000 Y:387.00000000 Total:29504.78000000
gkCActualNode: X:166.00000000 Y:391.00000000 Total:27227.50000000
gkCActualNode: X:168.00000000 Y:387.00000000 Total:29171.50000000

bestnode: X:170.00000000 Y:387.00000000 Total:26539.50000000
gkCActualNode: X:168.00000000 Y:385.00000000 Total:28948.30000000
gkCActualNode: X:170.00000000 Y:385.00000000 Total:28628.46000000

bestnode: X:166.00000000 Y:391.00000000 Total:27227.50000000
gkCActualNode: X:164.00000000 Y:389.00000000 Total:30104.14000000
gkCActualNode: X:164.00000000 Y:393.00000000 Total:27603.50000000

bestnode: X:164.00000000 Y:393.00000000 Total:27603.50000000
gkCActualNode: X:162.00000000 Y:391.00000000 Total:30730.38000000
gkCActualNode: X:162.00000000 Y:395.00000000 Total:27995.50000000

bestnode: X:162.00000000 Y:395.00000000 Total:27995.50000000
gkCActualNode: X:160.00000000 Y:393.00000000 Total:31383.50000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:160.00000000 Y:397.00000000 Total:28403.50000000
gkCActualNode: X:160.00000000 Y:395.00000000 Total:31668.46000000

bestnode: X:160.00000000 Y:397.00000000 Total:28403.50000000
gkCActualNode: X:158.00000000 Y:395.00000000 Total:32063.50000000
gkCActualNode: X:158.00000000 Y:399.00000000 Total:28827.50000000
gkCActualNode: X:158.00000000 Y:397.00000000 Total:32361.90000000

bestnode: X:170.00000000 Y:385.00000000 Total:28628.46000000
gkCActualNode: X:172.00000000 Y:383.00000000 Total:26099.50000000
gkCActualNode: X:168.00000000 Y:383.00000000 Total:28730.54000000
gkCActualNode: X:168.00000000 Y:387.00000000 Total:26739.50000000
gkCActualNode: X:170.00000000 Y:383.00000000 Total:28410.70000000

bestnode: X:172.00000000 Y:383.00000000 Total:26099.50000000
gkCActualNode: X:170.00000000 Y:381.00000000 Total:28214.38000000
gkCActualNode: X:172.00000000 Y:381.00000000 Total:27907.98000000

bestnode: X:168.00000000 Y:387.00000000 Total:26739.50000000
gkCActualNode: X:166.00000000 Y:385.00000000 Total:29289.58000000
gkCActualNode: X:166.00000000 Y:389.00000000 Total:27091.50000000

bestnode: X:166.00000000 Y:389.00000000 Total:27091.50000000
gkCActualNode: X:164.00000000 Y:387.00000000 Total:29875.50000000
gkCActualNode: X:164.00000000 Y:391.00000000 Total:27459.50000000

bestnode: X:164.00000000 Y:391.00000000 Total:27459.50000000
gkCActualNode: X:162.00000000 Y:389.00000000 Total:30488.30000000
gkCActualNode: X:162.00000000 Y:393.00000000 Total:27843.50000000

bestnode: X:162.00000000 Y:393.00000000 Total:27843.50000000
gkCActualNode: X:160.00000000 Y:391.00000000 Total:31127.98000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:160.00000000 Y:395.00000000 Total:28243.50000000

bestnode: X:172.00000000 Y:381.00000000 Total:27907.98000000
gkCActualNode: X:174.00000000 Y:379.00000000 Total:25699.50000000
gkCActualNode: X:170.00000000 Y:379.00000000 Total:28023.50000000
gkCActualNode: X:170.00000000 Y:383.00000000 Total:26291.50000000
gkCActualNode: X:172.00000000 Y:379.00000000 Total:27717.10000000

bestnode: X:174.00000000 Y:379.00000000 Total:25699.50000000
gkCActualNode: X:172.00000000 Y:377.00000000 Total:27547.66000000
gkCActualNode: X:174.00000000 Y:377.00000000 Total:27254.70000000

bestnode: X:170.00000000 Y:383.00000000 Total:26291.50000000
gkCActualNode: X:168.00000000 Y:381.00000000 Total:28542.22000000
gkCActualNode: X:168.00000000 Y:385.00000000 Total:26619.50000000

bestnode: X:168.00000000 Y:385.00000000 Total:26619.50000000
gkCActualNode: X:166.00000000 Y:383.00000000 Total:29087.82000000
gkCActualNode: X:166.00000000 Y:387.00000000 Total:26963.50000000

bestnode: X:166.00000000 Y:387.00000000 Total:26963.50000000
gkCActualNode: X:164.00000000 Y:385.00000000 Total:29660.30000000
gkCActualNode: X:164.00000000 Y:389.00000000 Total:27323.50000000

bestnode: X:174.00000000 Y:377.00000000 Total:27254.70000000
gkCActualNode: X:172.00000000 Y:375.00000000 Total:27383.66000000
gkCActualNode: X:172.00000000 Y:379.00000000 Total:25883.50000000
gkCActualNode: X:174.00000000 Y:375.00000000 Total:27090.70000000

bestnode: X:172.00000000 Y:379.00000000 Total:25883.50000000
gkCActualNode: X:170.00000000 Y:377.00000000 Total:27862.06000000
gkCActualNode: X:170.00000000 Y:381.00000000 Total:26187.50000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

bestnode: X:170.00000000 Y:381.00000000 Total:26187.50000000
gkCActualNode: X:168.00000000 Y:379.00000000 Total:28367.34000000
gkCActualNode: X:168.00000000 Y:383.00000000 Total:26507.50000000

bestnode: X:168.00000000 Y:383.00000000 Total:26507.50000000
gkCActualNode: X:166.00000000 Y:381.00000000 Total:28899.50000000
gkCActualNode: X:166.00000000 Y:385.00000000 Total:26843.50000000

bestnode: X:166.00000000 Y:385.00000000 Total:26843.50000000
gkCActualNode: X:164.00000000 Y:383.00000000 Total:29458.54000000
gkCActualNode: X:164.00000000 Y:387.00000000 Total:27195.50000000

bestnode: X:174.00000000 Y:375.00000000 Total:27090.70000000
gkCActualNode: X:176.00000000 Y:373.00000000 Total:25251.50000000
gkCActualNode: X:172.00000000 Y:373.00000000 Total:27233.10000000
gkCActualNode: X:172.00000000 Y:377.00000000 Total:25779.50000000
gkCActualNode: X:174.00000000 Y:373.00000000 Total:26940.14000000

bestnode: X:176.00000000 Y:373.00000000 Total:25251.50000000
gkCActualNode: X:178.00000000 Y:371.00000000 Total:25019.50000000
gkCActualNode: X:174.00000000 Y:371.00000000 Total:26811.02000000
gkCActualNode: X:176.00000000 Y:371.00000000 Total:26531.50000000

bestnode: X:178.00000000 Y:371.00000000 Total:25019.50000000
gkCActualNode: X:180.00000000 Y:369.00000000 Total:24803.50000000
gkCActualNode: X:176.00000000 Y:369.00000000 Total:26415.82000000
gkCActualNode: X:180.00000000 Y:371.00000000 Total:26028.78000000
gkCActualNode: X:178.00000000 Y:369.00000000 Total:26149.74000000

bestnode: X:180.00000000 Y:369.00000000 Total:24803.50000000
gkCActualNode: X:182.00000000 Y:367.00000000 Total:24603.50000000

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

gkCActualNode: X:178.00000000 Y:367.00000000 Total:26047.50000000
gkCActualNode: X:182.00000000 Y:371.00000000 Total:25805.58000000
gkCActualNode: X:182.00000000 Y:369.00000000 Total:25673.90000000
gkCActualNode: X:180.00000000 Y:367.00000000 Total:25794.86000000

bestnode: X:182.00000000 Y:367.00000000 Total:24603.50000000
gkCActualNode: X:184.00000000 Y:365.00000000 Total:24419.50000000
gkCActualNode: X:180.00000000 Y:365.00000000 Total:25706.06000000
gkCActualNode: X:184.00000000 Y:369.00000000 Total:25464.14000000
gkCActualNode: X:184.00000000 Y:367.00000000 Total:25345.90000000
gkCActualNode: X:182.00000000 Y:365.00000000 Total:25466.86000000

bestnode: X:184.00000000 Y:365.00000000 Total:24419.50000000
gkCActualNode: X:186.00000000 Y:363.00000000 Total:24251.50000000
gkCActualNode: X:182.00000000 Y:363.00000000 Total:25391.50000000
gkCActualNode: X:186.00000000 Y:367.00000000 Total:25149.58000000
gkCActualNode: X:186.00000000 Y:365.00000000 Total:25044.78000000
gkCActualNode: X:184.00000000 Y:363.00000000 Total:25165.74000000

bestnode: X:186.00000000 Y:363.00000000 Total:24251.50000000
gkCActualNode: X:188.00000000 Y:361.00000000 Total:24099.50000000
gkCActualNode: X:184.00000000 Y:361.00000000 Total:25103.82000000
gkCActualNode: X:188.00000000 Y:365.00000000 Total:24861.90000000
gkCActualNode: X:188.00000000 Y:363.00000000 Total:24770.54000000
gkCActualNode: X:186.00000000 Y:361.00000000 Total:24891.50000000

bestnode: X:188.00000000 Y:361.00000000 Total:24099.50000000
gkCActualNode: X:190.00000000 Y:359.00000000 Total:23963.50000000
gkCActualNode: X:186.00000000 Y:359.00000000 Total:24843.02000000
gkCActualNode: X:190.00000000 Y:363.00000000 Total:24601.10000000
gkCActualNode: X:190.00000000 Y:361.00000000 Total:23995.50000000
gkCActualNode: X:186.00000000 Y:361.00000000 Total:24219.50000000

gkCActualNode: X:188.00000000 Y:363.00000000 Total:24147.50000000
gkCActualNode: X:188.00000000 Y:359.00000000 Total:24644.14000000

bestnode: X:190.00000000 Y:359.00000000 Total:23963.50000000
gkCActualNode: X:192.00000000 Y:357.00000000 Total:23843.50000000
gkCActualNode: X:188.00000000 Y:357.00000000 Total:24609.10000000
gkCActualNode: X:192.00000000 Y:361.00000000 Total:24367.18000000
gkCActualNode: X:192.00000000 Y:359.00000000 Total:23867.50000000
gkCActualNode: X:188.00000000 Y:359.00000000 Total:24075.50000000
gkCActualNode: X:190.00000000 Y:357.00000000 Total:23939.50000000

bestnode: X:192.00000000 Y:357.00000000 Total:23843.50000000
gkCActualNode: X:194.00000000 Y:355.00000000 Total:23739.50000000
gkCActualNode: X:190.00000000 Y:355.00000000 Total:24402.06000000
gkCActualNode: X:194.00000000 Y:359.00000000 Total:24160.14000000
gkCActualNode: X:194.00000000 Y:357.00000000 Total:23755.50000000
gkCActualNode: X:192.00000000 Y:355.00000000 Total:23827.50000000

bestnode: X:194.00000000 Y:355.00000000 Total:23739.50000000
gkCActualNode: X:196.00000000 Y:353.00000000 Total:23651.50000000
gkCActualNode: X:192.00000000 Y:353.00000000 Total:23827.50000000
gkCActualNode: X:196.00000000 Y:357.00000000 Total:23683.50000000
gkCActualNode: X:196.00000000 Y:355.00000000 Total:23659.50000000
gkCActualNode: X:194.00000000 Y:353.00000000 Total:23731.50000000

bestnode: X:196.00000000 Y:353.00000000 Total:23651.50000000

Debug- Ausgabe endet hier, weil file voll war

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

11.0

Index

A

Action Steering....20
Actor....18
Actual Polygonal Floor 76
Agent....28, 90, 91, 93, 158
AI....*Siehe* Artificial Intelligence
Area 90
Areas....58, 97
Artificial Intelligence....9

Ä

ästhetischen Optimierung....62

B

belief/ desire/intention....17, 29, 31
best first search....46
Bézier Kurven....64, 67
binary heap 84
Black and White ..17, 27, 30, 31, 32, 34, 35
Breadth first search....42
Breitensuche....42, 43, 44, 45, 169

C

Catmull....64, 65, 66, 67, 122
Catmull Rom Formel....64, 65
C-cells....57
Charaktere....18
Charakteren....6
chasing....22
Closed.51, 53, 54, 61, 81, 83, 85, 111, 113, 147, 155, 156, 165, 170
Convex Polygons....57

D

Deep Blue 9
Dijkstra..25, 43, 44, 46, 47, 50, 88, 90, 169
Dreamcastkonsole 104

E

ELIZA.... 11
Engine 6, 58, 71, 72, 78, 86, 90, 95, 97, 100, 101, 104, 173
Entscheidungsbäumen 30, 33
Entscheidungsfindung 7, 17, 25, 26, 27, 29, 87
erkenntnistheoretischen Wahrscheinlichkeit 31
evading.... 22
Expertensystem.... 11

F

FIFO 42, 43, 45
First Person View 19, 20
*forced portal A**8, 118, 126, 129, 130, 134, 145, 157, 163, 164, 166, 169, 170
frame.... 21
frames per second 21
Fuzzy Logic 29, 30

G

Game Developer Conference..... 15, 16, 72
Gameplay.... 25
GAS ....*Siehe* - last portal search
generalized cylinders 58
Genrespiele 10, 15
GetSuccessor.... 125, 131, 157, 159
Gitternetz von Quadraten.... 57
goal area search.... 130, 144
Graphentheorie..... 25, 43, 46, 48, 49, 169

H

Handlungsplan.... 31
Hash Table 83, 85, 106, 165
Heapsortierung.... 135

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Heuristik .46, 47, 49, 50, 51, 55, 60, 61, 79, 80, 81, 85, 88, 89, 96, 116, 117, 118, 119, 121, 125, 126, 129, 130, 131, 135, 141, 164, 166, 170
hierarchische Pathfinding 68, 126
hierarchische Suche 63, 128, 129
Hindernisumgehungsstrategie 37

I

Ignorierung des Hindernisses.....57

J

Junktoren.....30

K

KI.....*Siehe* künstliche Intelligenz
Knotenbank 83, 93, 96, 111
Kollisionssystem 145
konvexe Polygone 57
künstlichen Intelligenz ..6, 9, 10, 11, 13, 14, 15, 21, 22, 175

L

last portal search.....130, 141
Learning opinions 33
Level.....75, 78, 81, 85, 164
Level Of Detail.....86
LIFO.....45
LOD..... *Siehe* Level Of Detail
lokale Pfadfindung 67
Lokation.....23
Lokationgrid ..93, 96, 98, 99, 100, 101, 117
Lokomotion.....7, 8, 20, 22, 94, 95, 167
LPS..... *Siehe* - last portal search

M

map representation.....59
maschinelles Beweisen.....10
Master Node List.....83
Mensch Computer Interaktion.....11
Mesh 57
Metzinger- Test.....13
Motivation 7, 8, 20, 25, 34, 87, 93, 94, 157, 158, 162, 167, 170
motor skills.....20
motorische Fähigkeiten.....20

N

Nachbarknoten... 42, 43, 51, 53, 59, 63, 64, 96, 98, 99, 132, 133, 153, 160
neuronalen Netze 13
Nodebank 96, 111, 147
Non Playing Characters 18
NPC.....*Siehe* non playing character

O

Objectmaps 23
Open. 42, 44, 46, 51, 53, 54, 61, 81, 83, 84, 85, 99, 104, 111, 113, 132, 147, 148, 151, 155, 165, 170
Orientation..... 55

P

Pattern..... 26
Pattern- KI 26
PC .....*Siehe* Playing Characters
Penalty..... 64, 119
Performanz- und Speicheroptimierung... 85
Perzeptron..... 30
Pfade..... 58
phänomenale Gestalt . 7, 12, 13, 16, 87, 91, 94, 125, 157, 167, 169, 170
phänomenalen Gehaltes..... 12
Physikmanagers..... 21
Playing Characters..... 18
Playstation 2..... 72
Points of visibility..... 57
Points of Visibility..... 78
Polygonal Floor Representation..... 77
portal to portal search..... 130, 134
potential functions 23, 24
PPA..... *Siehe* psychologischen Plausibilitätsagenten
PPTP .....Siehe - pure portal to portal search
priority queue..... 43, 44, 53, 84, 148, 170
PS2..... 72, 101
psychologischen Plausibilitätsagenten... 28
PTP .....*Siehe* - portal to portal search
pure portal to portal search... 125, 130, 131

Q

Quadratbaum..... 57
Quadratisches/ Sechseckiges Netz..... 75

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Quadtree 57
Quantoren 30

R

Rectangular grid 57, 75
Rectangular/ Hexagonal Grid 75
regelmäßige Vielecke 164, 165, 166
regular grid 64, 73, 82, 89, 91, 100, 121
Robotik 11
roundtables 16

S

Satz des Pythagoras ... 60, 79, 80, 116, 130, 137
Schnittpunktermittlung 137
Skript- KI 26
smooth 63, 89, 96, 148, 170
Sojourner 9
Space Invaders 26, 27
Spieledesign 27
Spieleprogrammierung 10
Steering 7, 8, 20, 22, 36, 71, 94, 95, 167
STL 104
straight 63, 89, 96, 125, 134, 140, 147, 170
Successor 98
Suchbereichsminimierung 75

T

Tasks 6, 31, 62
The Sims 16
Theoriematrix 29
Third Person View 19, 20
Tiefensuche 45
Turing Test 9

U

Umrissverfolgung 39, 40
universelle Pathfinding 67

V

Velocity 55
Verarbeitung natürlicher Sprache 11
Verfolgerrolle 22
Verfolgten 22
Vorlagenbasierte KI 26

W

Wertevorstellungen / Verlangen/ Intension Architektur 29, 34

X

XBox 72

Diplomarbeit
- Charakterverhalten in Computerspielen: Pathfinding -

Autor: René Greulich **Matrikelnr.: 110 18781**
Erstprüfer: Prof. Dr. H. Faeskorn – Woyke **Zweitprüfer: Prof. Dr. E. Ehses**

Erklärung:

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum Unterschrift