Informatik-Seminar SS07

Spiele KI

Thema: Hindernisnavigation

Sebastian Hammes mi2269

Inhaltsverzeichnis

Einleitung

Die Hindernisnavigation ist wahrscheinlich weitaus komplexer als man sich zuerst vorstellen wird. Das Gebiet der Hindernisnavigation deckt nicht nur das bloße Umlaufen von Hindernissen ab, denn diese Aufgabe bewältigt meistens schon ein Pathfindingalgorithmus. Vielmehr dreht es sich bei der Hindernisnavigation um das Beseitigen von Hindernissen, sodass neue Wege frei und damit benutzbar werden.

Zum Anfang erst einmal ein Beispiel aus dem täglichen Leben:
Stellen wir uns vor, wir sind Studenten an einer Fachhochschule, hören gerade eine Vorlesung und der Pausengong ertönt. Wir fassen den Entschluss zum Kaffeeautomaten zu gehen und uns einen Kaffee zu kaufen. Auf dem Weg dorthin werden wir um viele verschiedene Hindernisse navigieren müssen, um schließlich an unserem Ziel anzukommen. Das bloße Entlanglaufen eines Weges zum Kaffeeautomaten und dabei verschiedenen Hindernissen aus dem Weg zu gehen gehört auch zur Hindernisnavigation, ist aber noch nicht das Kernthema. Das eigentliche Kernthema finden wir in der Beseitigung von Hindernissen. Am Beispiel des Kaffeeautomaten wäre dies das Einstecken von 50 Cent für den Kaffee bevor man sich auf den Weg zum Kaffeeautomaten begibt.

An diesem Beispiel kann man sehr gut erkennen, worum es in der Hindernisnavigation geht. Das „Mitdenken“ und Vorausplanen um Hindernisse aus dem Weg räumen zu können ist die Hauptaufgabe der Hindernisnavigation. Am Ende dieser Ausarbeitung werden sie ein Konzept kennen lernen, dass genau dafür ausgerichtet ist. Dort wird es jedoch am Beispiel einer verschlossenen Tür mit Schalter näher gebracht.

Hindernisarten

Hindernisse können grob in drei Arten aufgeteilt werden. Zum einen sind dies die „unbeweglichen Hindernisse“. Zu dieser Art gehören z.B. Berge, Flüsse, Felsen, Mauern, Gebäude, etc. Diese Art von Hindernissen ist jedoch spielabhängig. Was diese Art von Hindernissen jedoch ausmacht ist, dass sie alle starre Hindernisse darstellen, die während des Spielverlaufs nicht verändert werden können. Zum anderen kann man sich auch „unbewegliche, beseitigbare Hindernisse“ vorstellen. Zu dieser Art gehören auch wieder je nach Spiel z.B. Bäume, Mauern, Gebäude, Brücken, Mienen, Türen, etc. Das wesentliche an dieser Hindernisart ist die Eigenschaft, dass sie während des Spielverlaufs verändert oder beseitigt werden können. Türen können geöffnet werden, Mauern können eingerissen werden. Diese Art von Hindernissen stellt den Hauptbestandteil bei der Hindernisnavigation dar. Die dritte und letzte Art sind die „beweglichen Hindernisse“. Ein paar Beispiele zu dieser Art könnten Aufzüge, bewegliche Steine/Kisten, aber auch eigene oder gegnerische Einheiten sein, die ihre Position im Spielverlauf verändern können.

[Bild 1-3] ScreenShots aus Command&Conquer

Wo findet man Hindernisnavigation

Der menschliche Spieler

Je nach Spielgenre muss ein menschlicher Spieler mehr oder weniger von der Hindernisnavigation übernehmen. Sehen wir uns einmal die First-Person-Spiele an. Hierbei steuert der Spieler eine virtuelle Person durch eine 3D-Szene. Hierzu muss er alle Arten von Hindernissen selbst bewältigen. Er muss seinen gespielten Charakter um unbewegliche Hindernisse bewegen, muss beweglichen Hindernissen ausweichen und muss schließlich auch unbewegliche, beseitigbare Hindernisse zerstören oder aus dem Weg räumen. Sehen wir uns dazu im Gegensatz einmal die Strategiespiele an. Hier bekommt der menschliche Spieler von einer künstlichen Intelligenz Unterstützung. Für das Umlaufen von unbeweglichen und beweglichen Hindernissen wird dem menschlichen Spieler ein Pathfindingalgorithmus zur Verfügung gestellt, sodass er sich nicht mehr darum kümmern muss, wie er z.B. seine Einheiten von A nach B bekommt ohne auf Hindernisse zu treffen. Die dritte Art von Hindernissen, die unbeweglichen, beseitigbaren Hindernisse, jedoch muss er selbst bewältigen. Soll heißen, dass der menschliche Spieler an dieser Stelle selbst entscheiden muss, wie er ein unbewegliches, beseitigbares Hindernis umgeht oder beseitigt. Dies ist vor allem darin begründet, dass unbewegliche, beseitigbare Hindernisse auch von strategischem Vorteil sein können, wenn man sie nicht beseitigt, sondern diese umgeht. Diese Entscheidung kann und darf eine KI dem menschlichen Spieler dabei auch nicht abnehmen.

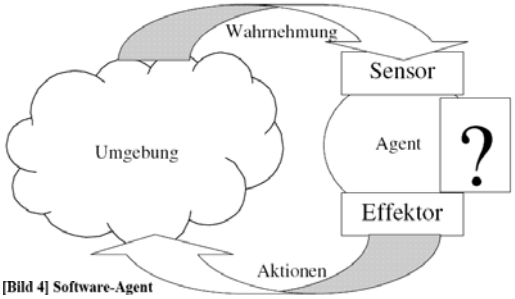
Der Software-Agent

Definition

„Als Software-Agent bezeichnet man ein Computerprogramm, das weitgehend unabhängig von Benutzereingriffen arbeitet, es löst Aktionen aufgrund eigener Initiative aus (proaktiv), reagiert auf Änderung der Umgebung (reaktiv), es kommuniziert mit anderen Agenten und lernt aufgrund zuvor getätigter Entscheidungen bzw. Beobachtungen.“ [WIKI:SA]

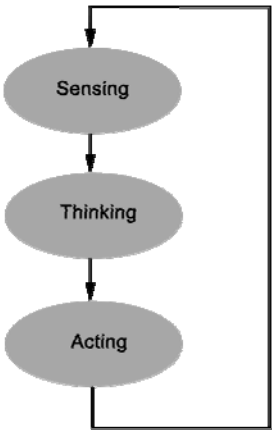
Grundaufbau

Der Software-Agent ersetzt also einen z.B. fehlenden menschlichen Gegenspieler. Egal wie komplex und „intelligent“ ein Software-Agent programmiert ist, seine arbeitsweise ändert sich dabei wenn überhaupt nur gering. Zur Veranschaulichung soll das Bild Nummer 4 dienen. Die Wolke stellt das Spielgeschehen mit all seinen Facetten dar, welche der Software-Agent über verschiedene Sensoren wahrnehmen kann. Ein Software-Agent verarbeitet anschließend die gewonnenen Informationen intern und trifft dabei Entscheidungen, welche schließlich über den Effektor in Aktionen enden, die wiederum Einfluss auf die Umgebung nehmen.



[Bild 4] Software-Agent

Schaut man sich einmal den inneren Kreislauf des Agenten an, so kann man Parallelen zum menschlichen Verhalten erkennen. Ein Mensch versucht über verschiedene Sensoren seine Umwelt wahrzunehmen, diese Informationen auszuwerten und schließlich auf diese Informationen mit Aktionen zu reagieren. So arbeitet auch der Software-Agent von innen. Zuerst führt er ein Sensing aus. In dieser Phase ermittelt er alle ihm zur Verfügung stehenden Informationen über den aktuellen Spielverlauf/-stand. In der zweiten Phase, dem Thinking, wertet der Software-Agent seine gewonnenen Informationen aus und ermittelt auf dieser Grundlage einen Plan. Dieser Plan ist bis dorthin noch abstrakt, soll heißen, dass nur das geplante Ziel feststeht. Die konkrete Umsetzung dieses Plans aber noch offen ist. Ein Beispiel hierfür könnte sein, dass der Software-Agent aufgrund der aktuellen Spiellage seine Einheiten von einer Position zu einer anderen bewegen möchte. Welche konkreten Teilaufgaben dabei auf die Einheiten zukommen, bleiben hier erst einmal unberücksichtigt. Die dritte und Abschließende Phase bildet das Acting. Hier wird ein konkreter Plan ausgeführt. Ein Beispiel hierfür ist das Abarbeiten einzelner, kleiner Aufgaben, um das geplante Ziel zu erreichen. [WIKI:SA] [UNI:TRIER]



[Bild 5] Software-Agent von innen

Hindernisnavigation beim Software-Agenten

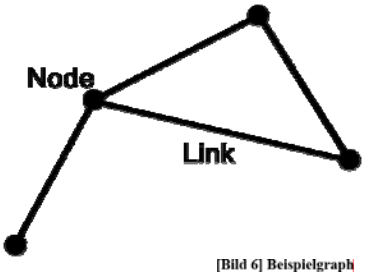
Wo also nun ist die Hindernisnavigation im Kreislauf des Software-Agenten anzusiedeln? Das Thinking hat einen abstrakten Plan als Resultat, jedoch für die Umsetzung braucht das Acting einen konkreten Plan. Die Hindernisnavigation ist also genau zwischen dem Thinking und dem Acting anzusiedeln. Die Aufgabe der Hindernisnavigation ist es, einen abstrakten Plan in einen konkreten Plan umzusetzen und dies so, dass das Erreichen des Ziels in viele kleine Unterschritte aufgeteilt wird. Bei der Unterteilung werden die unterschiedlichen Hindernisarten berücksichtigt und gesondert verarbeitet. Ein anschauliches Beispiel hierzu ist am Ende dieser Ausarbeitung zu finden.

Graphen

Diese Art der Weltdarstellung findet vor allem in 3D-Szenen seine Anwendung. Dies ist nicht nur auf seine Flexibilität zurückzuführen, sondern auch auf die effiziente CPU- und Memorynutzung beim Navigieren auf dem Graphen. Jedoch begleitet die Graphen auch eine unangenehme negative Seite. Die Erzeugung und Optimierung der Graphen nimmt sehr viel Zeit und Arbeit in Anspruch. Dies liegt vor allem daran, dass Graphen nicht von Computern optimal generiert werden können und Level-Designer immer wieder diese nachbessern und korrigieren müssen.

Aufbau eines Graphen

Ein Graph besteht aus zwei wesentlichen Bestandteilen. Den so genannten Nodes, welche in der Weltdarstellung Punkte im Raum darstellen und den zwischen den Nodes liegenden Links, die die Wege zwischen zwei Nodes darstellen. Daher kann man einen Graphen auch als Nodemap bezeichnen. Grundsätzlich sind Links zunächst einmal richtungsunabhängig, sodass Wege zwischen zwei Nodes immer in beide Richtungen benutzt werden können.

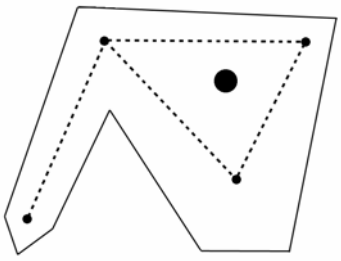


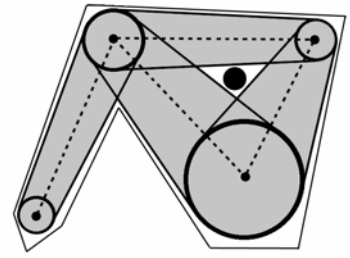
[Bild 6] Beispielgraph

Schließlich gibt es noch den Begriff des Paths, der einen Weg zwischen hintereinander liegenden, verbundenen Nodes bezeichnet. Der Grundaufbau eines Graphen kann der bildlichen Darstellung auf der rechten Seite entnommen werden. [TORONTO]

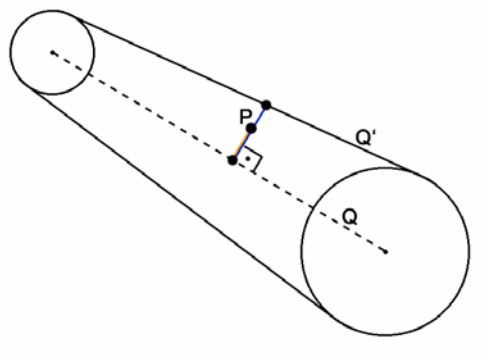
Probleme für die Hindernisnavigation

Ein bloßer Graph ist zunächst einmal nur recht schwer für die Hindernisnavigation zu gebrauchen, denn Nodes beschreiben genaue Punkte im Raum und Links schmale, gradlinige Wege zwischen diesen. Für die Hindernisnavigation wird allerdings eine etwas genauere Darstellung benötigt, um Hindernisse abbilden zu können. Ein kleiner Pfeiler in einem Raum würde, wie in der Abbildung Nummer 7 zu sehen, ein viel größeren Platz als Hindernis markieren, als dies nötig wäre. Zieht man so einen einfachen Graphen zur Hindernisnavigation heran, so ist die Umsetzung der Hindernisnavigation später sehr unrealistisch, denn der Software-Agent könnte sich nur exakt auf den Links bewegen. Dies ist jedoch nicht das Ziel der Hindernisnavigation. Um dieses Problem zu umgehen muss der Graph um eine Angabe erweitert werden, welche den Graphen so flexibel macht, dass unbewegliche Hindernisse besser dargestellt werden können. Hierzu führt man das Prinzip der Radien ein. Jeder Node wird um die Angabe eines Radius erweitert, der den Freiraum um diesen Node definiert. Verbindet man gleichzeitig noch benachbarte Nodes tangential an den Radien, so wird der Freiraum um Links automatisch mit definiert. Die Erweiterung der Nodes um einen Radius ist sehr effizient hinsichtlich der Datenmenge, benötigt jedoch einen hohen Rechenaufwand, um die optimalen Radien festzulegen und ein hohes Maß an menschlicher Nachbearbeitung, um einen perfekten Graphen zu erhalten. Die Erweiterung des Graphen um Radien lässt schließlich eine recht genaue Darstellung der Welt inklusive unbeweglicher Hindernisse dar, welches eine Grundvoraussetzung für die Hindernisnavigation ist. [AI GP]


[Bild 7] Graph in einem Raum mit Hindernis


[Bild 8] um Radien erweiterter Graph

Bezeichnung des „Irrens“



[Bild 9] Berechnung des Irrens

Unter der Bezeichnung des „Irrens“ wird verstanden, wie weit sich ein beliebiger Charakter bereits vom Ursprünglichen Weg hin zu einem unbeweglichen Hindernis bewegt hat. Dies ist vor allem wichtig, um einen Charakter sicher um ein Hindernis navigieren zu können. Das Bild rechts stellt einen Ausschnitt aus einem Graphen dar, der aus zwei Nodes, dem Link zwischen diesen, den Radien um die Nodes und den tangentialen Radiusverbindungen besteht. Zudem ist ein Punkt P eingezeichnet, der die aktuelle Position eines Charakters darstellen soll, der auf dem Weg von einem zum anderen Node ist. Wichtig für diesen Charakter ist es nun, zu jeder Zeit bestimmt zu können, wie weit er sich bereits vom direkten Weg (Q) wegbewegt hat und wie viel Freiraum sich noch bis zur Korridorgrenze (Q’) befindet. Denn würde der Charakter diesen Korridor verlassen, so würde er mit hoher Wahrscheinlichkeit mit einem unbeweglichen Hindernis kollidieren. Die Berechnung, wie weit ein Charakter bereits geirrt ist, stellt sich dabei recht einfach dar:
Zuerst wird der Abstand des Charakters zum direkten Link berechnet, anschließen wird an der aktuellen Position die Breite des Korridors berechnet:

Abstand von P zu Q = a	Distanz des Charakters zum direkten Link
Abstand von Q’ zu Q = b	Breite des Korridors an der aktuellen Stelle

Aus diesen beiden Werten kann ein Charakter genau bestimmen, wie weit er bereits vom direkten Weg abgekommen ist, wie viel Freiraum er noch bis zur Korridorgrenze, also bis zu einem unbeweglichen Hindernis, hat und kann entsprechende Aktionen z.B. zum Korrigieren der Laufrichtung einleiten. [AI GP]

Kommentierung des Graphen

Für die Hindernisnavigation ist ein erweiterter Graph jedoch immer noch nicht ganz zu gebrauchen, denn unbewegliche, beseitigbare Hindernisse können damit noch nicht abgebildet werden. Hierzu muss der Graph um weitere Informationen ergänzt werden.
Besondere Punkte im Raum können mit Flags versehen werden und somit für den Software-Agenten markiert werden. Diese Markierungen von Nodes dienen vor allem zum markieren von besonders guten strategischen Punkten, Schaltern, die Türen öffnen können, etc. Aber auch Links können mit Markierungen versehen sein. Die Möglichkeit Links zu markieren wird vor allen benötigt, um unbewegliche, beseitigbare Hindernisse darstellen zu können. Ein unbewegliches, beseitigbares Hindernis könnte z.B. eine verschlossene Tür darstellen, die erst durch das Drücken eines Schalters geöffnet werden kann. Hierzu wird der Link, der durch diese Tür führt, als „closed“ markiert und diese Markierung wird erst dann aufgehoben, wenn der Schalter gedrückt wird. Es ist also möglich durch die Kommentierung der Links Wege als temporär unpassierbar zu markieren, was der Darstellung von unbeweglichen, beseitigbaren Hindernissen gerecht wird. Zudem können noch weitere Eigenschaften an Links angehängt werden wie z.B. den aktuellen Zustand des Charakters, wenn er diesen Link passieren will (geduckt, an Klippe hängend, etc.). Aber auch das Vermerken von Beziehungen zu anderen Nodes ist möglich. Hier wäre als Beispiel das Vermerken des SchalterNodes an dem Link anzuführen, der durch eine Tür geht. Zudem wird es möglich, die zuvor richtungsunabhängigen Links durch Setzen von Flags richtungsabhängig zu gestalten. Dies wird vor allem dann wichtig, wenn Links nur noch in einer Richtung verfolgt werden können. Dies könnte z.B. beim Herunterrutschen einer Stange der Fall sein. [AI GP]

Hindernisdarstellung mit Graphen

Unbewegliche Hindernisse

Abhängig von der Qualität des Graphen können unbewegliche Hindernisse recht gut Abgebildet werden. Dazu werden sie beim Errichten des Graphen einfach ausgelassen, soll heißen, Nodes und Links umschließen ein unbewegliches Hindernis so, dass ein Freiraum zwischen Nodes und Links entsteht. Um diesen Freiraum möglichst genau den Hindernisformen anpassen zu können, werden die Nodes um eine Radiusangabe erweitert, was zur folge hat, dass unbewegliche Hindernisse sehr gut dargestellt werden können. Die Informationsmenge, um diese Art von Hindernissen darstellen zu können verhält sich dabei sehr Ressourcen sparend.

Unbewegliche, beseitigbare Hindernisse

Diese Art von Hindernissen wird in einem Graphen anders berücksichtigt, denn bei unbeweglichen, beseitigbaren Hindernissen ist es möglich, dass das Hindernis beseitigt werden kann und somit ein zuvor versperrter Weg frei werden muss. Würden unbewegliche, beseitigbare Hindernisse wie unbewegliche Hindernisse dargestellt, so wäre eine Anpassung der Graphen nach Beseitigung notwendig. Zudem wäre es erst gar nicht möglich solche Hindernisse zu beseitigen, denn der Weg zu diesem Hindernis wäre erst gar nicht in dem Graphen verzeichnet. Daher werden unbewegliche, beseitigbare Hindernisse anders dargestellt. Sie werden durch Links abgebildet, denen durch Flags bestimmte Bedingungen angehängt werden können, sodass ein Link z.B. erst dann benutzbar wird, wenn man zuvor einen Schalter gedrückt hat. Bei dieser Art der Hindernisdarstellung kommt es darauf an, dass die Kommentierung des Graphen sauber durchgeführt wird. So ist es also möglich einen Link durch z.B. eine geschlossene Tür darzustellen, aber ihn für die Benutzung zunächst einmal zu sperren. Hierfür wird z.B. der Link durch eine Tür mit dem Flag „closed“ versehen. Erst wenn dieses Flag aufgehoben wird, dies kann durch verschiedene Aktionen geschehen, wird die Tür als passierbar markiert und der Software-Agent kann schließlich diesen Link benutzen. Unbewegliche, beseitigbare Hindernisse können also recht einfach in das Konzept der Graphen aufgenommen werden. Hierzu werden grundsätzlich einmal alle Links in den Graphen aufgenommen, aber zunächst unpassierbare Links werden durch Flags markiert, sodass erst dieses Hindernis beseitigt werden muss, um den Link benutzen zu können. Diese Art der Hindernisdarstellung ist wiederum sehr Ressourcen sparend.

Bewegliche Hindernisse

Diese Art von Hindernissen wird in dem Konzept der Graphen nicht berücksichtigt. Würden bewegliche Hindernisse jedoch berücksichtigt, hätte dies zur folge, dass bei jeder Bewegung eines beweglichen Hindernisses der Graph angepasst werden müsste. Dies ist jedoch kaum zu bewältigen, denn wie weiter oben bereits beschrieben, ist es nur sehr schwer, einen „perfekten“ Graphen zu erstellen und dies als Voraussetzung die menschliche Nachbesserungsarbeit hat. Bei der Anpassung für bewegliche Hindernisse müsste dies jedoch vom Computer übernommen werden, was zur Folge hätte, dass zuvor “perfekte“ Graphen verschlechtert werden. Zudem wäre es vorstellbar, dass jede Art von Charakter seinen eigenen Graphen besitzt, abhängig von seinen speziellen Fähigkeiten. Dies würde dazu führen, dass nicht nur ein, sondern viele Graphen angepasst werden müssten und dafür ist der Aufwand einfach zu hoch. Daher muss sich bei der Hindernisnavigation mit einem anderen Konzept beholfen werden. Dies könnte zum Beispiel das Konzept der Boundingboxes/-spheres sein, die zur Laufzeit Kollisionen mit beweglichen Hindernissen erkennen und diese abwenden können. Bewegliche Hindernisse werden also nicht im Konzept der Graphen berücksichtigt.

Goal-based Path Planning und Execution

Jeder Software-Agent ermittelt während des Thinkings einen abstrakten Plan, den er im Acting umsetzt. Jedoch stellt sich da die Frage, wie setzt man einen abstrakten Plan in einen konkreten Plan um, um diesen dann ausführen zu können? Dazu gibt es das Konzept des Goal-based Path Planning, welches ausschließlich dazu dient, einen abstrakten Plan in einen konkreten umzusetzen, sodass dieser schließlich ausgeführt werden kann. Der Konzeptname beinhaltet das Wort Goal = Ziel und dies ist auch die Grundlage dieses Konzeptes. Ein abstrakter Plan wird in viele kleine Teilziele zerlegt, die schließlich elementare Schritte darstellen, die ein Charakter ausführen kann. Als Beispiel könnte hier der abstrakte Plan sein, dass ein Charakter einen Fußball wegschießen möchte. Bei dem Goal-based Path Planning würde dieses Ziel in kleinere Ziele unterteilt. Diese könnten sein: das Laufen zum Ball, das Ausholen und schließlich das Schießen des Balles. Aus einem abstrakten Plan wurde mit Hilfe des Goal-based Path Planning ein konkreter Plan erzeugt, der dann von einem Charakter ausgeführt werden kann. [AI GP] [UNI:ULM]

Eigenschaften von Goals

Es gibt verschiedene Arten von Goals, die für verschiedene Problemstellungen von Nutzen sind, jedoch besitzen diese alle ein paar gemeinsame Eigenschaften. Grundvoraussetzung für das Goal-based Path Planning ist es, dass Goals ihre zu bewältigenden Aufgaben wieder weiterreichen können. Dazu ist es möglich, dass ein Goal so genannte Subgoals erstellen kann, die es nach und nach abarbeiten kann um das Gesamtziel, das es als Goal erreichen soll, zu erreichen. Goals haben also die Möglichkeit, ihre Aufgabe auf kleinere Subgoals zu verteilen. Bei dieser Vorgehensweise ist es wichtig, dass ein Goal sein Parentgoal über Erfolg oder Misserfolg unterrichten kann. Dazu besitzt jedes Goal ein Flag, dass bei erfolgreicher Abarbeitung des Goals auf true oder bei Misserfolg auf false gesetzt wird. Anhand dieses Flags kann das Parentgoal erkennen, ob die Abarbeitung seiner Subgoals wie geplant verläuft oder ob dabei Probleme auftreten. Bei Problemen ist es dann die Aufgabe des Parentgoals, seine Subgoals neu zu planen oder den Misserfolg weiter in der Goalhierarchie nach oben zu geben. Als dritte wichtige Eigenschaft gelten die Kosten, die ein Goal bei der Abarbeitung verursacht. Anhand dieser Kostenfunktionen wird es ermöglicht, dass der Aufwand verschiedener Goals in Relation zueinander gesetzt werden können und somit sinnvoll bei der Subgoalerstellung zwischen verschiedenen Möglichkeiten abgewägt werden kann. [AI-CENTER]

Goal-Arten

Zur Unterteilung eines Plans in viele kleine Teilziele sind verschiedene Goals nötig. Die wichtigsten werden im Folgenden erklärt. [AI GP]

Goal_GotoPosition

[Bild 10] Darstellung von GotoPosition

Dieses Goal ist das höchste in der Hierarchie der Goals. Dieses Goal kann nicht als Subgoal eines anderen Goals verwendet werden. Goal_GotoPosition wird als Parameter ein Punkt übergeben, der irgendwo innerhalb eines Graphen liegen muss, jedoch nicht genau auf einem definierten Node. Goal_GotoPosition dient zum Erreichen eines Zielpunktes irgendwo innerhalb eines Graphen. Dazu stellt es selbst elementare Funktionen bereit die dazu dienen, vom nächstgelegenen Node des Zielpunktes zum Zielpunkt zu gelagen. Die Aufgabe, wie man zu diesem nächstgelegenen Node kommt, gibt es weiter ab an Subgoals.

```
Goal_GotoPosition (ZielPosition) {
  /* ... Subgoals ... */
  /* elementare Funktionen zum Erreichen der ZielPosition
     vom nächstgelegenen Node der ZielPosition */
}
```

Goal_GotoNode

[Bild 11] Darstellung von GotoNode

Goal_GotoNode wird als Parameter ein Node übergeben, der sich innerhalb des Graphen befinden muss. Die einzige Aufgabe dieses Goals ist es, einen Charakter z.B. von seinem ursprünglichen Punkt zu dem ZielNode zu bewegen. Dazu kann dieses Goal seine Arbeit auch wieder in kleinere Teilabschnitte zerlegen, die bereits genannten Subgoals. Dieses Goal besitzt keine eigenen Funktionen, denn es verteilt seine Aufgabe ausschließlich auf Subgoals.

```
Goal_GotoNode (ZielNode) {
  /* ... Subgoals ... */
}
```

Goal_FollowLink

[Bild 12] Darstellung von FollowLink

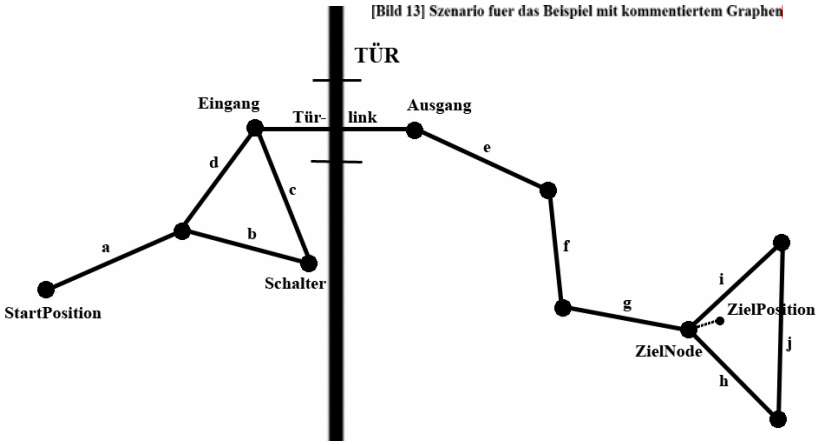
Dieses Goal ist in der Hierarchie der Goals am untersten Ende angesiedelt. Es kann seine Arbeit nicht mehr weiter auf Subgoals aufspalten und besitzt daher nur elementare Funktionen zum erfolgreichen abarbeiten des Goals. Goal_FollowLink wird als Parameter ein Link übergeben und dient ausschließlich für das erreichen eines benachbarten Nodes über den entsprechenden Link.

```
Goal_FollowLink (Link) {
  /* elementare Funktionen zum überwinden eines Links */
}
```

Beispiel anhand des Goal-based Path Planning

Als verdeutlichendes Beispiel soll schließlich eine zusammenhängende Aufgabe mit Hilfe des Goal-based Path Plannings gelöst werden. Nehmen wir einmal an, wir haben folgendes Szenario abgebildet durch den Graphen im Bild. Das Bild stellt eine Szene mit 11 Links und 10 Nodes dar. Alle benötigten Nodes und Links sind bereits kommentiert. Anhand der Kommentierung kann man erkennen, dass sich in diesem Szenario ein unbewegliches, beseitigbares Hindernis befindet, die Tür, die nur geöffnet werden kann, wenn der Schalter gedrückt wird. Die Startposition ist bereits eingezeichnet. Ein Software-Agent hat schließlich als abstrakten Plan beschlossen, dass er als nächstes zur Position ZielPosition möchte. Nun ist es die Aufgabe des Goal-based Path Plannings den abstrakten Plan so umzusetzen, dass er zu einem konkreten Plan mit vielen kleinen Teilzielen wird. Um überhaupt erst einmal einen Weg von der Startposition zur Zielposition zu finden muss ein Pathfindingalgorithmus bemüht werden. Dabei ist zu beachten, dass das Pathfinding zunächst einmal nicht die unbeweglichen, beseitigbaren Hindernisse berücksichtigen darf. Dieser würde folgenden Weg vorschlagen:

[Bild 13] Szenario fuer das Beispiel mit kommentiertem Graphen



Startposition -> a –> d –> TürLink –> e –> f –> g –> ZielPosition

Beim Goal-based Path Planning dient dieser Weg nun als Grundlage, um den konkreten Plan aufzubauen. Dabei wird der Plan rückwärts aufgebaut, da so sich in den Weg stellende Hindernisse flexibel beseitigt werden können, denn erst beim Treffen auf das beseitigbare Hindernis kann entschieden werden, welche zusätzlichen Aktionen geplant werden müssen. Würde der Plan vorwärts aufgebaut, so würde man erst bei der Tür bemerken, dass zuvor noch ein Schalter gedrückt werden muss, was zu Umwegen und Schwierigkeiten führen kann. [AI GP]

Aufbau des konkreten Plans mit Teilzielen

Aufgabe ist es nun, einen konkreten Plan zu entwickeln. Dafür werden wir zuerst das Goal_GotoPosition bemühen müssen, denn dies ist in der Hierarchie der Goals an oberster Stelle angesiedelt. Dazu übergeben wir diesem Goal unsere ZielPosition. Goal_GotoPosition wird nun als nächstes den nächstliegenden Node von ZielPosition ermitteln und dabei den ZielNode finden. Da es nicht die Aufgabe selbst von Goal_GotoPosition ist, einen Plan zum ZielNode zu erstellen, so gibt es diese Aufgabe an ein Subgoal weiter. Dazu erstellt er ein Subgoal namens Goal_GotoNode und übergibt diesem als Parameter den Wert ZielNode. Aufgabe von Goal_GotoNode(ZielNode) ist es nun, einen konkreten Plan zu entwickeln um den Charakter nach ZielNode zu bewegen. Dafür unterteilt auch Goal_GotoNode(ZielNode) seine Aufgabe in 4 Teile und gibt diese an Subgoals weiter. Zuerst werden 3 Subgoals von Goal_FollowLink erstellt, denen jeweils ein Link als Parameter übergeben wird (Goal_FollowLink(g), Goal_FollowLink(f), Goal_FollowLink(e)). Als 4. Subgoal wird Goal_GotoNode ein Goal erstellen müssen, das einen konkreten Plan erstellen kann, welches den Charakter zum Schalter bewegt, den Schalter drückt und dann durch die Tür laufen lässt. Genau dafür ist das Goal_GoThroughDoor geeignet. Diesem Goal wird als Parameter der Link der Tür mitgegeben, sodass es daraus einen kompletten, konkreten Plan für das Erreichen und Durchlaufen der Tür erstellen kann. Aber auch Goal_GoThroughDoor(Türlink) wird seine Arbeit auf 3 Subgoals auslagern. Zum einen auf das Goal_FollowLink(TürLink), welches das Durchlaufen der Tür zur folge hat. Diesem Subgoal wird noch ein anderes Goal vorgeschaltet, das so genannte Goal_GotoNode(Eingang), welches den konkreten Plan zum Erreichen des Eingangs ermittelt. Dieses wird dann schließlich seine Arbeit an Goal_FollowLink(c) abgeben. Goal_GotoNode(Eingang) wird ein drittes Goal vorgeschaltet, das so genannte Goal_HitSwitch(Schalter). Dieses Goal wird den konkreten Plan erstellen, wie der Charakter zum Schalter gelangt und diesen drücken wird. Goal_HitSwitch(Schalter) wird als letztes die elementare Funktion PushButton() des Schalters ausführen und damit die Tür öffnen. Dem wird jedoch ein Subgoal Goal_TurnTowards(Schalter) vorangestellt, welches bewirkt, dass der Charakter sich direkt vor den Schalter stellt. Diesem voraus geht das Goal_GotoNode(Schalter), um einen konkreten Plan zu entwickeln, wie man den Schalter erreichen kann. Diese Aufgabe wird dann wieder in zwei Teile zerlegt, einmal in das Goal_FollowLink(a) und Goal_FollowLink(b).

Diese Umsetzung eines abstrakten Plans in einen konkreten Plan mit Hilfe des Goal-based Path Planning dient schließlich dem Software-Agenten für das Acting. Im Acting kann er diesen konkreten Plan Schritt für Schritt abarbeiten und wird so sein zuvor geplantes Ziel erreichen. Dies nennt man auch Path Execution.

Konkreter Plan

Hier sieht man den konkreten Plan in seiner hierarchischen Struktur und kann sehr gut erkennen, welche Goals welche Subgoals erstellt haben. Bei der Abarbeitung des konkreten Plans würde man nun von oben beginnen.

```
Goal_GotoPosition(ZielPosition) {

  Goal_GotoNode(ZielNode) {

    Goal_GoThroughDoor(Türlink) {

      Goal_HitSwitch(Schalter) {

        Goal_GotoNode(Schalter) {

          Goal_FollowLink(a) {}

          Goal_FollowLink(b) {}

        }

        Goal_TurnTowards(Schalter) {}

        PushButton(); // Elementare Funktion des Schalters die aufgerufen wird, KEIN Goal

      }

      Goal_GotoNode(Eingang) {

        Goal_FollowLink(c) {}

      }

      Goal_FollowLink(TürLink) {}

    }

    Goal_FollowLink(e) {}

    Goal_FollowLink(f) {}

    Goal_FollowLink(g) {}

  }

}
```

Abschluss

Ich hoffe einen guten Einblick in das komplizierte Thema der Hindernisnavigation mit seinen vielen Facetten geben zu können. Das abschließende motivierende Beispiel sollte schlussendlich viele Unklarheiten beseitigt haben, denn anhand eines Beispieles lassen sich oft viele Dinge einfacher darstellen und auch verstehen.

Anhang

Literaturverweis

[AI GP] AI Game Programming Wisdom Band 1, Steve Rabin

[WIKI:SA] http://de.wikipedia.org/wiki/Software-Agent

[UNI:TRIER] http://strathisla.uni-trier.de/lectblog/wp-content/14KIPrint.pdf

[TORONTO] http://www.cs.toronto.edu/~ccollins/publications/Graph_Visualization_talk.pdf

[UNI:ULM] http://medien.informatik.uni-ulm.de/lehre/courses/ws0304/odc/Berichte/KI-StateOfTheArt.pdf

[AI-CENTER] www.ai-center.com/events/gdc-2003-roundtable/GDC2003.ppt

Bildverweis

[Bild 1-3]: http://www.cnc-inside.de/galerie.php?go=kategorie&kat=609
[Bild 4]: http://www.is.informatik.uni-duisburg.de/wiki/images/f/fa/BestandteileAgent.GIF
[Bild 5]: Eigenproduktion
[Bild 6]: Eigenproduktion
[Bild 7]: Eigenproduktion
[Bild 8]: Eigenproduktion
[Bild 9]: Eigenproduktion
[Bild 10]: Eigenproduktion
[Bild 11]: Eigenproduktion
[Bild 12]: Eigenproduktion
[Bild 13]: Eigenproduktion