Methoden der Künstlichen Intelligenz für Computerspiele auf Basis semantischer Modelle interaktiver VR/AR-Systeme

Anton Feldmann[1] Marc Erich Latoschik[2]

[1]Postfach 10 01 31, Universität Bielefeld, Technische Fakultät, AG Wissensbasierte Systeme, 33615 Bielefeld, Tel:+49 (0) 521 7807497, EMail: anton.feldmann@uni-bielefeld.de
[2]Treskowallee 8, FHTW Berlin, FB 4, Internationale Medieninformatik, 10318 Berlin, Tel:+49 (0) 30 5019-0, Email: marc.latoschik@fhtw-berlin.de

Abstract: Methoden der Künstlichen Intelligenz (KI) nehmen einen zentralen Stellenwert in der Entwicklung heutiger Computerspiele ein. Eine explizite lose Koppelung isolierter KI-Module mit benötigten Modulen zur graphischen, physikalischen und auditiven Simulation etc. wird zusehends komplex und stellt eine kritische Schnittstelle innerhalb interaktiver Systeme dar. Dieses Paper dokumentiert eine alternative eng gekoppelte Modellierung einer integrierenden Architektur basierend auf einer generellen semantischen Abstraktionsschicht. Ein semantisches Netz führt in der entworfenen Architektur die verschiedenartigen Simulationsmodule zusammen und stellt die Grundlage für ein einheitliches Entwurfsparadigma "Semantic Reflection" dar. Dieses findet hier für die Realisierung verschiedener benötigter KI-Verfahren im Computerspielekontext Anwendung. Am Beispiel der Pfadplanung, der Entscheidungsfindung, der Verhaltenssteuerung, des Lernens sowie der Anbindung eines Scriptinginferfaces werden die Vorteile dieses Ansatzes illustriert.

Stichworte: Computerspiele, Künstliche Intelligenz, semantisches Netz, Semantic Reflection, scripting

1 Einleitung

Computerspiele stellen zusehends einen wichtigen Wirtschaftsfaktor dar und sind ebenfalls im Fokus aktueller Forschungprojekte. Moderne Computerspiele bedienen sich einer Fülle von Algorithmen aus der Computergraphik, der Soundgenerierung, der physikalischen Simulation sowie der Künstlichen Intelligenz (KI) etc. KI-Methoden nehmen dabei einen zusehend zentralen Stellenwert ein. Die Einbettung dieser Verfahren erweist sich als komplex, da sie auf vielfältige Weise mit der Funktionalität und dem Prozessfluss der Module für die Spielgraphik, die Physik und die Soundgenerierung etc. verknüpft sind. Die Einbettung der KI sowie die Umsetzung der eigentlichen KI-Methoden benötigt dabei einen hohen Entwicklungsaufwand.
Grundlage der hier vorgestellten Entwicklung eines KI-Kernels stellt ein Softwareentwicklungsparadigma für das Design intelligenter interaktiver VR/AR-Systeme dar: "Semantic

Reflection" [LatFrWe06] beschreibt das Zusammenspiel verschiedener Softwarekomponenten auf einer semantischen Ebene. Es erweitert das informatische Reflektionsprinzip um ontologische Bindungen und ermöglicht eine einheitliche Modellierung von low-level Softwarefunktionen bis hin zu abstrakten Welt- und Verhaltensbeschreibungen.
"Semantic Reflection" wurde in Vorarbeiten im Kontext interaktiver VR/AR-Systeme entwickelt [HeuSchiLat05, LatSchi03]. Es ist Basis eines semantischen Simulationskerns (SCIVE -Simulation Core for Intelligent Virtual Environments) für intelligente interaktive Systeme. SCIVE ermöglicht einen Softwareentwurf mittels semantischer Modellierung. Ein "Functional Extendable Semantic Network (FESN)"[LatSchi03] stellt die Basisrepräsentationsschicht bereit, um "Semantic Reflection" als einheitliches Prinzip auf den unterschiedlichen vertikalen Ebenen im Softwareentwurf und der Implementierung zu realisieren.
Im Folgenden wird die Anwendung dieses Softwareentwurfsparadigmas auf die Domäne prototypischer KI-Methoden in Computerspielen – welche eine hohe softwaretechnische Ähnlichkeit zu VR/AR-Systemen aufweisen – beschrieben. Zu den implementierten Methoden zählen:

1. Algorithmen zur Pfadplanung (path planning)
2. Entscheidungsbäume (decision trees)
3. Zielorientiertes Verhalten (goal-oriented behavior)
4. Neuronale Netze (neural networks)
5. Skriptsprachenanbindung (scripting)

Das scripting spielt in der Arbeit eine gesonderte Rolle. Durch das scripting wird eine Schnittstelle für andere Programmiersprachen bereitgestellt, um mit den Objekten des FESN zu interagieren. Die Realisierung der fünf verschiedenen Methoden erfolgt in einem einheitlichen Muster mittels so genannter Traverser direkt auf der semantischen Repräsentation, welche ihrerseits durch den Simulationskernel direkt mit den Datenrepräsentationen der anderen Simulationsmodule synchronisiert wird und so jederzeit den aktuell gültigen konsistenten Weltzustand reflektiert.

2 Repräsentation im Semantischen Netz

Ein semantisches Netz besteht aus einer endlichen Menge von Knoten und einer endlichen Anzahl von Kanten, welche die Knoten miteinander verbinden. Die formale Grundlage dieser Repräsentationsform sind die so genannten Beschreibungslogiken (Description Logics–DLs). Die DLs sind eine Untermenge der Prädikatenlogik der 1. Stufe (PL/1). Im Gegensatz zu der PL/1 sind DLs entscheidbar, sie verzichten im Allgemeinen etwa auf die Quantoren oder Funktionen der PL/1.

Auf diesem Netzwerk können qualitative sowie quantitative Wissensinhalte niedergelegt werden. Erstere in der Regel durch Beziehungen (Relationen) zwischen Objekten (den Konzepten). Letztere durch zusätzliche Wertattributierung an den Konzepten (s. folgende Ausführungen). Somit kann mittels des semantischen Netzes der Spielkontext und -zustand aller Objekte repräsentiert werden.

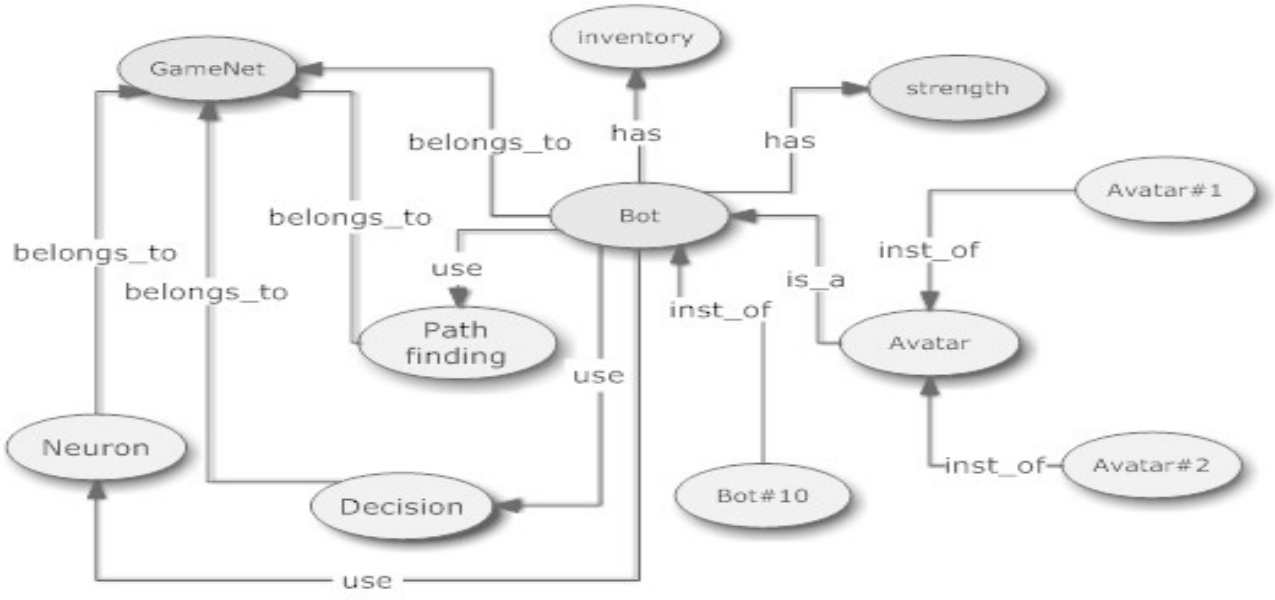

Abbildung 1: Beispiel eines semantischen Netzes aus dem Kontext der Computerspiele

Um die Anforderungen interaktiver intelligenter VR/AR-Systeme und hier speziell der Computerspiele zu realisieren, erfuhr das zugrundeliegende FESN vielfältige Erweiterungen. So wurden die Knoten durch das Konzept der Frames und Slots [Min74] erweitert. Als Wissensrepräsentationsmethode dienen Frames der Assoziation verschiedener Attribut-Wert Paare zu Konzepten. Durch gezielte Vererbung dieser Attribute in so genannten Frame-Slots werden Frames häufig zur Modellierung von Proto- und Stereotypen eingesetzt. Die Erweiterung der Netzknoten zu Frames erlaubt es, zusätzliche Nutzdaten mittels Slots an den entsprechenden Knoten zu speichern. Die Slots können so konkrete Daten wie numerische Werte, Strings, oder auch 3D Graphikvertices etc. speichern.
Abbildung 1 illustriert ein semantischs Netz für die Repräsentation der Zusammenhänge zwischen verschiedenen Objekten eines typischen Computerspiels. Die Eigenschaften eines so genannten Bots, eines durch den Computer simulierten Mit- oder Gegenpielers (abgeleitet von Robot), sind durch die Relationen “has“ und ”use” verdeutlicht. Vererbungshierarchien sind durch die Relationen “inst_of” und “is_a” verdeutlicht, wobei die “inst_of” Relationen auf eine Instanz des Objektes verweist und die “is_a” Relation ein Objekt des Types verdeutlicht. “belongs_to” beschreibt eine räumliche Nähe zu dem Objekt, zu der diese Relation verweist. Im Abbildung 1 wird z.B. der Bot dem GameNet mittels der “belongs_to” Relation zugewiesen.
Im selben Schritt, ist das FESN durch Domänen (domains) erweitert worden. Domänen

gruppieren Teilbereiche der Netze zu sinntragenden Gruppen. Abbildung 2 stellt solch eine Gruppierung dar. Als Erweiterung des FESN werden Domänen selber auf dem FESN als Knoten repräsentiert. Mit ihnen kann man Aussagen über bestimmte Cluster präpositionaler Wissenstrukturen ausdrücken. Die durchgeführten Änderungen erweitern die Aussagefähigkeit der DLs und beinhalten dadurch auch Risiken in Bezug auf die Entscheidbarkeit und Schlussfolgerungskomponenten. Durch eine strikte Verwendung der getroffenen Erweiterungen durch spezialisierte Inferenzmechanismen, etwa der Domänen zur Wissenszuordnung und -modularisierung, wird diesem Umstand Rechnung getragen.

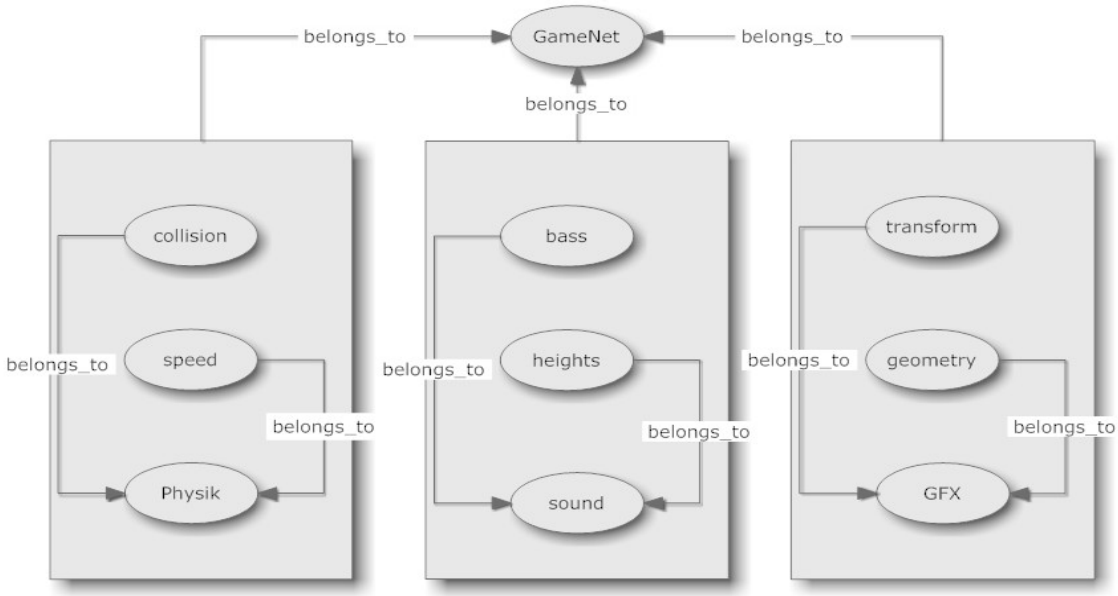

Abbildung 2: Beispiel eines FESN mit Domänen. Die Domänen sind durch Rechtecke dargestellt. Die "belongs_to" Relationen zu den rechteckigen Knoten verdeutlichen, die Bedeutung von Domänen im Kontext.

Vorteil der gewählten strukturellen Wissensrepräsentation ist die leichte Verständlichkeit des gültigen Anwendungskontextes. Netze können aus etlichen tausenden von Knoten bestehen, doch ist es für den Bentuzer des FESN durch die selektive Darstellung bestimmter Domänen schnell ersichtlich, welcher Kontext hier betrachtet wird. Durch die Bereitstellung eines XML- basierten persistenten Streamingformats (SNIL-Semantic Network Interface Language), kann der aktuelle Zustand des semantischen Netzes gesichert werden [LatSchi03]. Dies ermöglicht Lade- und Speicherungsoptionen aller Zustände innerhalb eines Computerspiels.

2.1 Erweiterung des FESN

Die im Rahmen dieser Arbeit notwendigen Erweiterungen werden im Folgenden konkret erläutert. Die erste Modifikation betrifft das Objektmodell des FESN, welches um eine gemeinsame Oberklasse (`SNObject`) sowohl für Konzepte als auch für Relationen erweitert wurde. Das zugrundeliegende Klassendiagramm wird in Abbildung 3 verdeutlicht.

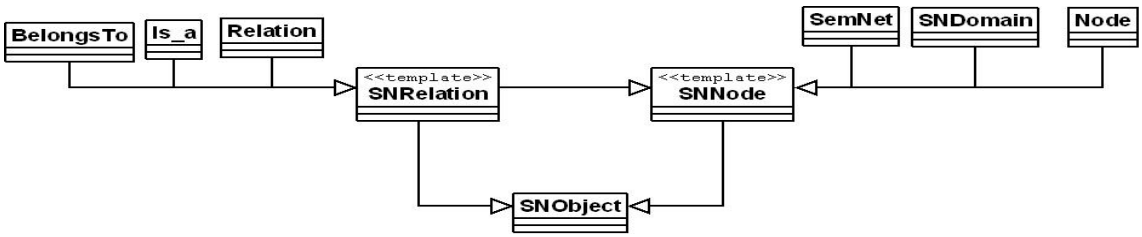

Abbildung 3: UML Vererbungsdiagramm für das FESN

Als Konsequenz können Relationen zwischen Relationen gebildet werden. Relationen erhalten den Status von Objekten erster Ordnung (den Konzepten im semantischen Netz). Dies ist begründet in der Notwendigkeit, Relationen Slots zuzuordnen und sie verschiedenen Teilnetzen zuzuordnen (s. Abbildung 4). Das hierdurch umstrukturierte FESN hat nun zwei unterschiedliche Arten von Relationen:

1. Meta-Relationen und

2. Relationen

Die gepunkteten Relationen in Abbildung 4 stellen die Meta-Relation "belongs_to" dar und die Meta-Relation "is_a" des FESN ist dem gegenüber gestrichelt dargestellt. Die Meta–Relationen geben Auskunft über die Zugehörigkeit oder den Typ der Objekte des FESN selbst. So ist der Typ einer Relation durch seine Meta-Relationen zu einem Knoten des Netzes, der den entsprechenden Typen darstellt, bestimmt. In Abbildung 4 wird dies durch den Knoten "is_a" angedeutet.

2.2 Scripting

Scripting ist eine etablierte informatische Lösung, um komplexe Programme mit einer einfachen interaktiven Schnittstelle zu erweitern. Dies ermöglicht sowohl die Wiederverwendung, als auch eine einfache Parametrisierung und Anpassung an neue Domänen und Einsatzbereiche. Eine Erweiterung des FESN um ein generelles Scripting Interface stellt diese Eigenschaften auch für das semantische Netz zur Verfügung. Für jeden der im Abschnitt 3 erwähnten Traverser steht ein Interface bereit, damit von einer Skriptsprache aus die entsprechenden Objekte des FESN manipuliert werden können.
Zur weitgehenden Unabhängigkeit von dedizierten Skriptsprachen wurde das Interface auf Basis des SWIG-Projektes erstellt, welches die Anbindung vieler unterschiedlicher Sprachen zur Compilezeit unterstützt.
Das Scripting Interface für das FESN wurde so implementiert, dass sich viel der Funktionalität aus C++ im Interface wiederspiegelt. Dazu wurden die gewünschten Funktionen mit in das SWIG File aufgenommen.

```
module semnet
%{
```

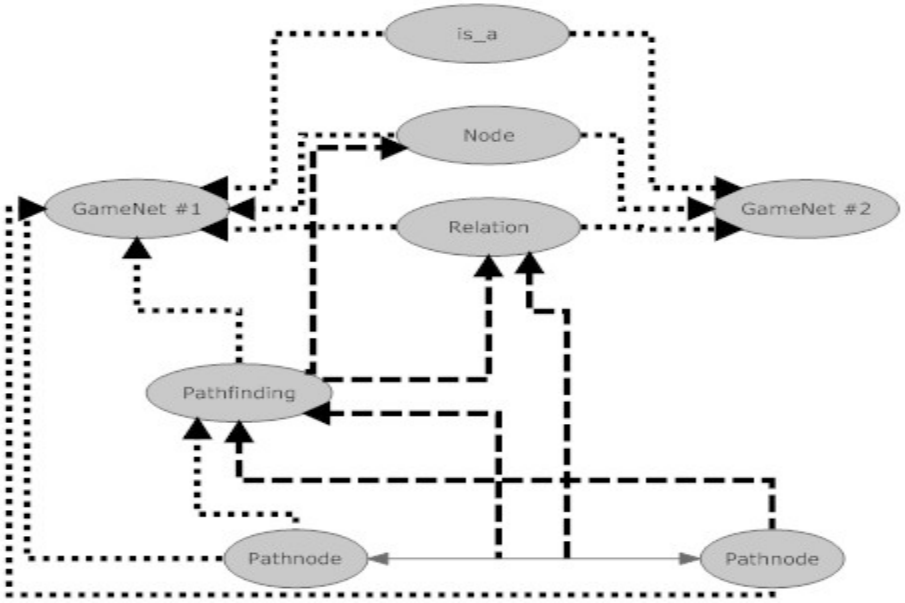

Abbildung 4: Die Struktur des FESN wurde dahingehend erweitert, dass alle Objekte des FESN Knoten sind. Die Meta– Relation “belongs_to” ist durch eine gestrichelte Relation dargestellt, wohingegen die “is_a“ Meta– Relation durch eine gepunkteten Relation verdeutlicht ist.

```
 #include "../include/semnet.h"
 using namespace semnet;
%}

class SemNet{
 addNode(Node node);
 ...
}
```

3 Traverser

Die gewählte Graphstruktur der semantischen Repräsentationsschicht ermöglicht als Konsequenz eine vereinheitlichte Umsetzung der verschiedenen Inferenzprozeduren auf Basis so genannter ”Semantischer Traverser”. Im Gegensatz zu Besuchsmustern (visitor pattern) aus dem OOP-Design (OOP–Object Oriented Programming) muss die besuchte Struktur dabei keine Akzeptormethoden bereitstellen. Sämtliche benötigte Information für die Traverser sind durch die semantische Information in den Knoten und Relationen selbst gegeben. Die dadurch resultierende Typunabhängigkeit ermöglicht eine einfache und abstrahierende Umsetzung auch in streng getypten Sprachen wie C++, welches aus Geschwindigkeitsgründen für die Low-level Implementierung der Umsetzung der KI-Methoden vorzugsweise Verwendung fand.

3.1 Pathfinding

Die Aufgabe des Pathfinding ist es, zwischen zwei gegebenen Orten, in der Regel kodiert durch Knoten einer Graph-basierten Verbindungsrepräsentation, entweder überhaupt eine Verbindung, oder eine "beste" Verbindung zu finden. Im Spielekontext dominant ist die Suche in einem gewichteten Graphen nach Übergängen, die von Knoten 1 zu Knoten 2 führen (s. Abbildung 5). Pathfinding findet vielfältige Anwendung bei der Steuerung autonomer Agenten, Bots, oder auch teilautonomer Einheiten innerhalb der Spielwelten.

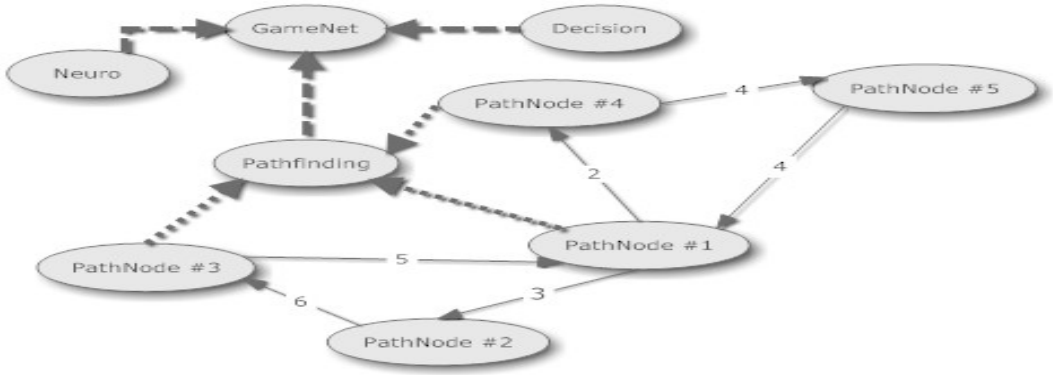

Abbildung 5: Die Gewichte des Graphen sind durch Zahlen an den Relationen verdeutlicht. Im Fall des Pathfinding zeigen diese Zahlen die Kosten zum Wechseln von einem zum nächsten Knoten an.

Zwei klassische Varianten für Suchalgorithmen auf solchen Graphen sind der Dijkstra- und der A*-Algorithmus (s. [Mil06, RusNor03, Dij59]). Als Erweiterung des Dijkstra-Algorithmus verwendet der A* nicht nur die Kosten bis zur aktuellen Position $g(n)$, sondern zusätzlich eine Heuristik $h(n)$. $h(n)$ schätzt für jede explorierte Position die Kosten bis zum Ziel. Setzt man $h(n) = 0$ degeneriert der A* zum Dijkstra–Algorithmus.
Die Graphrepräsentation des FESN führt direkt zu einer Speicherung der Kosten als Relationenattribute oder -slots. Die Traverser lesen dieses Datum aus, indem sie den Typ des Slots über die semantische Annotation ermitteln und in ihren internen Heuristikberechnungen einsetzen. Die Traverser verwenden dabei eine Prioritätswarteschlange, welche im Algorithmus 1 und Algorithmus 2 mit `open` bezeichnet wird. Im Algorithmus 1 wird der Zugriff auf die Kosteninformation über die Funktion $d(x,y)$ implementiert, welche die bisherigen Gesamtkosten d über jeden Traverserschritt aufsummiert.

Die Traverser werden mittels `applyTo` auf den Startknoten `node` eines Netzes `semnet` bis zu einem Abbruchknoten `endnode` angewendet. Der A*-Traverser (s. Algorithmus 2) verwendet neben der Kostenzugriffsfunktion $d(x,y)$ die Heuristik $h(x, \mathtt{e}ndnode)$ zur Schätzung der Kosten bis zum Ziel `endnode`.

Algorithmus 1 : Dijkstra-Traverser()

Input : get start node (node), get base FESN (semnet), get end node (endnode)
Output : get FESN with path (semnet)

```
1  Algorithmus :applyTo(node, semnet, endnode)
2  begin
3      g ⟵ 0
4      open ⟵ {(node, g)}
5      while open ≠ ∅ do
6          current ⟵ first element of open
7          if current = endnode then
8              return success
9          end
10         else
11             foreach c_i ⟵ Relations of current do
12                 s ⟵ node connected with current through c_i
13                 g ⟵ g + d(current, s)
14                 if (s ∉ open) ∧ (costOf(s) <= g) then
15                     open ⟵ s
16                 end
17             end
18         end
19     end
20 end
```

3.2 Decision-Tree

Decision-Trees sind einfache, aber häufig eingesetzte Strukturen zum Repräsentieren von Entscheidungsverläufen in interaktiven Systemen und Computerspielen. Knoten eines Baumes stehen hier für konkretisierbare Zustände innerhalb der Spielewelten und der Agentenprozesse. Die Kanten repräsentieren die Bedingungen zum Wechsel in andere Zustände. Eine Übertragung dieser Repräsentationsform auf das semantische Netz kann direkt erfolgen. In Abbildung 6 wird ein Ausschnitt eines Decision-Trees gezeigt. Die Abbildung illustriert die Entscheidungsmodellierung eines Bots unter Berücksichtigung der unterschiedlichen Ereignisse innerhalb eines Spiels.

Der Algorithmus 3 beschreibt den von uns implementierten Decision-Tree-Traverser Algorithmus. Es werden die Attributwerte der Knoten mit den Attributwerten der von dem Knoten ausgehenden Kanten verglichen. Sollten die Werte übereinstimmen, so wird im Knoten s die Methode $run()$ ausgeführt. Die $run()$ Methode startet in dem Knoten s den vorgegebenen Traverser.

Algorithmus 2 : AStar-Traverser()

Input : get start node (node), get base FESN (semnet), get end node (endnode), heuristic(h)

Output : get FESN with path (semnet)

```
Algorithmus :applyTo(node, semnet, endnode)
begin
    g ⟵ 0
    open ⟵ {(node, g, h)}
    while open ≠ ∅ do
        current ⟵ first element of open
        if current = endnode then
            return success
        end
        else
            foreach c_i ⟵ Relations of current do
                s ⟵ node connected with current through c_i
                g ⟵ g + d(current, s) + h(s, endnode)
                if (s ∉ open) ∧ (costOf(s) <= g) then
                    open ⟵ s
                end
            end
        end
    end
end
```

Algorithmus 3 : Decision-Tree()

Input : get start node (node), get base FESN (semnet)

Output : get FESN (semnet)

```
Algorithmus :applyTo(node, semnet)
begin
    foreach c_i ⟵ Relations of node do
        s ⟵ node connected with current through c_i
        if Value of node = Value of s then
            s ⟶ run()
        end
        else
            continue
        end
    end
end
```

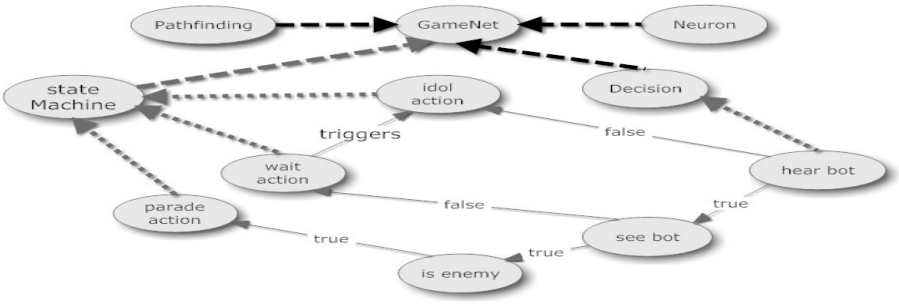

Abbildung 6: Ein Decision-Tree in dem die Aktionen über die Relation “trigger” ausgelösst werden.

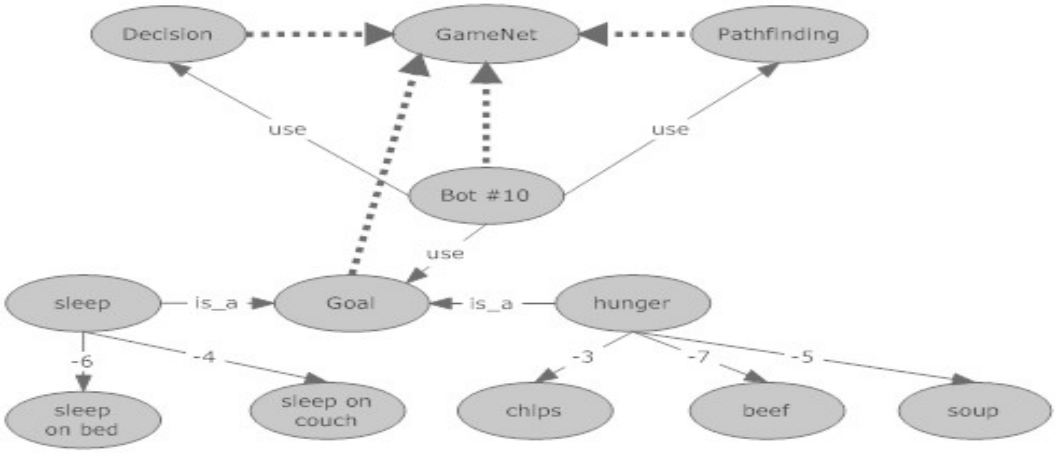

Abbildung 7: Goal-Tree eingebettet in das Game FESN.

3.3 Goal-Oriented Behavior

Goal-Oriented Behavior ist eine Verhaltensteuerung auf Basis einer Zielbeschreibung. Analog zu den bereits illustrierten Verfahren können auch diese Zielbeschreibungen in einer Graphstruktur von Plänen (möglichen Pfaden) zur Zielerreichung modelliert werden. Die Lösung des Problems besteht wiederum in einer Suche nach dem geeigneten Plan, um ein bestimmtes Ziel zu erreichen. Zur Realisierung zeigen sich hier zwei der Stärken des Ansatzes der semantischen Modellierung, Abstraktion und Wiederverwendbarkeit. So kann auf den bereits entwickelten Dijkstra-Traverser zurückgegriffen werden, ohne spezielle Anpassungen vorzunehmen. Als Ergebnis wird ein partielles FESN zurückgegeben, welches den detaillierten semantisch annotierten Plan zum Erreichen des Ziels repräsentiert.
In Abbildung 7 wird gezeigt, wie der “Bot #10” sich für eine der zwei möglichen Ziele entscheiden kann. Zum einem kann der Bot schlafen oder essen. Nun gibt das Computerspiel durch die State-Machine den Zustand des Bots wieder, der dann eine Aussage über das nächste Ziel des Bots schlußfolgert. Sollte der Bot hungrig sein, dann wird er essen, ist er müde so wird der Bot schlafen.

Algorithmus 4 : Multilayer-Perzeptron()

Input : get start node (node), get base FESN (semnet)

Output : get FESN with the learned weights (semnet)

```
1  Algorithmus :applyTo(node, semnet)
2  begin
3      input ⟵ FESN
4      hidden ⟵ FESN
5      output ⟵ FESN
6      foreach c_i ⟵ {hidden, output} do
7          c_i ⟶ feedforward
8      end
9      foreach c_i ⟵ {output, hidden} do
10         error ⟵ geterror
11         c_i ⟶ adjustWeights(error)
12     end
13 end
```

3.4 Neuronale Netze

Eine besondere Form von KI-Methoden befasst sich mit dem Lernen. Lernverfahren haben erst in wenigen aktuellen Spielen Einsatz gefunden, da das dynamische Verhalten lernender Systeme im Vorfeld häufig nicht eindeutig analysierbar ist und im Endprodukt zu unerwünschtem Verhalten führen kann. Es ist dennoch eine Methode, welcher ein hohes Potential eingeräumt wird, um computersimulierte Agenten mit glaubwürdigerem, da an Umstände und Benutzer angepasstem, Verhalten zu versehen.
Wir haben das Multilayer Perzeptron (MLP) eingesetzt, um das Lernen in Computerspielen zu realisieren. Ein MLP ist eine in Schichten angeortnete Menge von Neuronen. Der MLP-Traverser, gibt als Resultat ein FESN zurück. Dieses FESN enthält das gelernte MLP.
In Algorithmus 4 ist dargestellt, wie das von uns implementierte Multilayer Perzeptron (MLP) gelernt wird. Die erste Schleife arbeitet den Weg durch das Netz von Anfang bis Ende. Dies geschieht mittels der Funktion $feedforward$. Am Ende angekommen, wird der resultierende Klassifikationsfehler $error$ mittels der Funktion $adjustWeights(error)$ rückwärts durch das MLP propagiert.

4 Fazit

Eine semantische Modellierung benötigter KI–Funktionalitäten im Spielekontext hat sich als vielversprechend erwiesen. Besonders hervorzuheben ist die strukturelle Nähe vielfältiger KI-Methoden und -Strukturen zur bereitgestellten Basisrepräsentation eines erweiterten semantischen Netzes. Dieses ermöglicht eine einheitliche Modellierung der KI-Methoden im

Zusammenspiel mit den verschiedenen Simulationskomponenten. Auf eine genauere Erläuterung letzterer wurde aus Platzgründen verzichtet und im Stand der Forschung auf die Vorarbeiten verwiesen.
Die hier gemachten Erfahrungen haben uns erneut bestätigt, dass ein Großteil informatischer Methoden mit Hilfe einer Graphrepräsentation und spezialisierten Traverserkonzepten realisierbar sind.
Die erfolgte Umsetzung der KI-Komponente hat gezeigt, dass sich das ursprünglich ausgearbeitete "Semantic Reflection" Paradigma auch auf den Kontext typischer KI-Methoden zur Pfadplanung, zur Entscheidungs- und Zielfindung, zum Lernen sowie zum Scripting in Computerspielen einsetzen lässt. Speziell diese Umsetzung hat dabei benötigte generelle Erweiterungen an der Basisrepräsentation des Simulationskerns identifiziert. Diese Erweiterungen haben den zugrunde liegenden Simulationskern entscheidend um Ausdrucksstärke hin zu einer allgemeinen Modellierungsmöglichkeit für verschiedenste Anwendungsfelder bereichert.

Literatur

[Dij59] Dijkstra, E. W.: A note on two problems in connexion with graphs. In: *Numerische Mathematik* 1 (1959), S. 269–271.

[HeuSchiLat05] Heumer, G., Schilling, M., Latoschik, M. E.: Automatic Data Exchange and Synchronization for Knowledge-Based Intelligent Virtual Environments. In: *Proceedings of the IEEE VR2005*. Bonn, Germany, 2005, S. 43-50.

[LatFrWe06] Latoschik, M. E., Fröhlich, C., Wendler, A.: Scene Synchronization in Close Coupled World Representations using SCIVE. In: *The International Journal of Virtual Reality* 5 (2006), Nr. 3, S. 47–52.

[LatSchi03] Latoschik, M. E., Schilling, M.: Incorporating VR Databases into AI Knowledge Representations: A Framework for Intelligent Graphics Applications. In: *Proceedings of the Sixth International Conference on Computer Graphics and Imaging*, ACTA Press, 2003, S. 79–84.

[Mil06] Millington, I.: *Artificial Intelligence For Games*. Morgan Kaufmann, 2006.

[Min74] Minsky, M.: A Framework for Representing Knowledge. AI Memo 306, MIT AI Laboratory, Cambridge, 1974.

[RusNor03] Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approch*. 2. Prentice Hall, 2003.