Wissensrepräsentation
Vorlesung

Sommersemester 2008

3. Sitzung

Dozent

Nino Simunic M.A.

Computerlinguistik, Campus DU

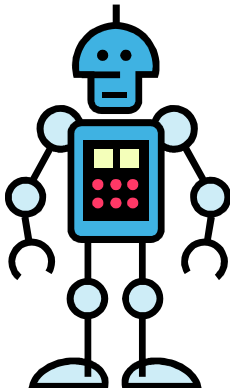
UNIVERSITÄT DUISBURG ESSEN

- **Roboter (Planungprozess)**
  - **Robotik, STRIPS**
- Agenten (in Computerspielen)
  - FSM, Agenten-Architektur, PEAS
  - *Utility function*

Roboter: Definitionsversuch

Ein autonomer, »intelligenter« **Roboter** soll eine **Aufgabe** in einer **bestimmten Zeit** und in einer **Umgebung** ausführen, die er selber wahrnehmen kann.

Ein Roboter ist ein komplexes System aus folgenden Teilsystemen



Effektoren (i.d.R. Gliedmaßen-artiges)

Sensoren (Drucksensoren, Kameras, Mikrofone, …)

Steuer- und Regeleinheiten (Rechner, Programme, …)

(Zusatzausrüstung (Werkzeuge, Teilegeber,…))

Robotik: Definitionsversuch

»Extremstes« Gebiet der KI: Alle Schwerpunkte spielen eine wesentliche Rolle

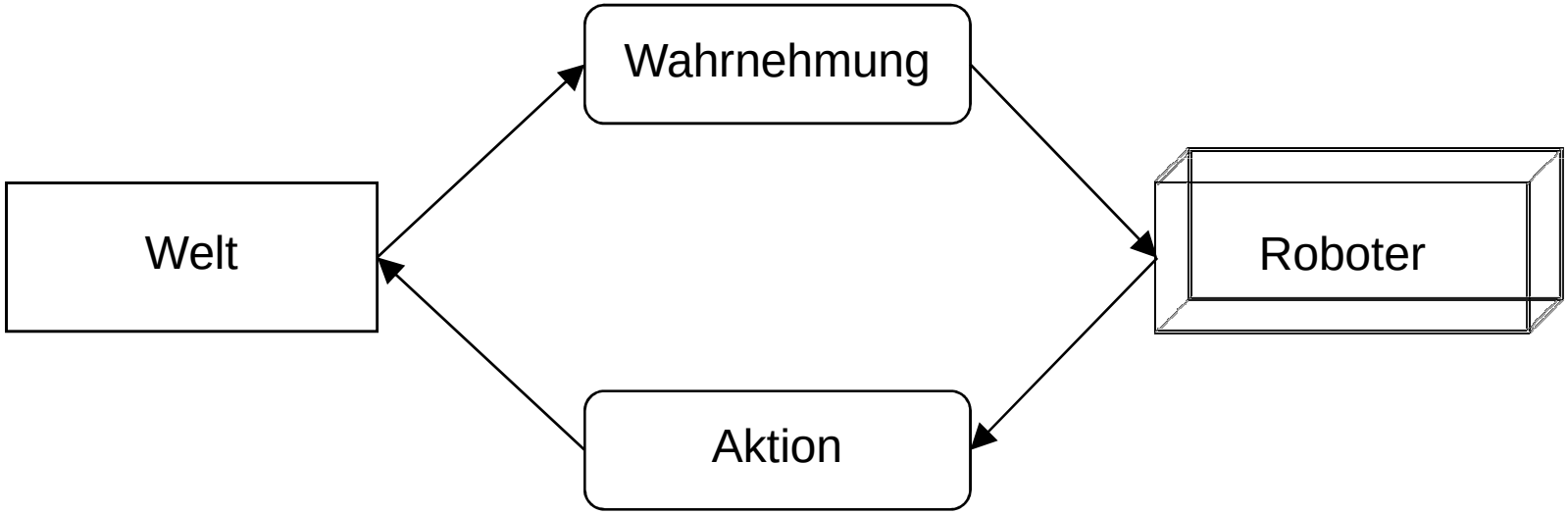
Methode: Integration von KI-Teilbereichen

- Repräsentation von Wissen
- Such- und Inferenzprozesse
- Plangenerierungsverfahren
- Bildverstehen
- Verstehen natürlicher (gesprochener) Sprache
- KI-Programmierung
- Lernverfahren, …

Ziel: Intelligenter, autonomer Roboter.

- Wichtig: Die Welt bleibt so, wie sie ist, und der Roboter agiert in ihr an.

Wie funktioniert ein Roboter, was macht er?



Ziel(e): Aktion (der Aktuatoren/Effektoren)

Z.B. Aufheben des Werkzeugs, Bewegung des Beins, …

Lösungsprinzip einer Aufgabe

»Arnibot, bringe Bücher AB, CD und EF in Raum 364!«

→ **Planen einer komplexen Aktion**

- Suche nach dem besten Weg
- Aufheben und Verstauen von Gegenständen

…

→**Teilaktionen und Teilzielen**

→ **Überprüfung** und **Revision** des Plans

…

Teilaktionen und -ziele

Komplexe Aktion: »*Arnibot, bringe Bücher AB, … in Raum 364!*«

Teilziele/-aktionen

1. Wo sind die Bücher AB, CD, und EF?

   Raum 136

2. Arnibot (Raum 118)  Arnibot (Raum 136)

3. Sind die Bücher wirklich dort?

   ja: Geeigneten Transportbehälter finden …

   nein: Ort der Bücher herausfinden …

4. Arnibot (Raum 136)  Arnibot (Raum 136)

…

Umweltmodell

- Ein Roboter (Agent) muss ein möglichst **genaues Umweltmodell** besitzen, um **in der Umwelt erfolgreich** zu **agieren**
  - Umweltmodell muss intern repräsentiert werden, u.a. um effizient **planen** zu können.
  - Im Gegensatz zu vielen anderen Modellen in der Informatik und verwandten Gebieten.

Unsicherheitsfaktoren

- Jedes Werkstück ist anders
  - Produktion mit Toleranz
  - Auch Effektoren sind Werkstücke mit Toleranz
- Sensor-Auswertung ergibt meist mit Unsicherheit behaftete Information
- Minimierung von Unsicherheiten bei Industrierobotern:
  - Anpassung der Welt an die Roboterfähigkeiten
  - → Robotergerechte Gestaltung industrieller Arbeitsplätze.

Der Planungsprozess: Zutaten

- Eine Repräsentation des **initialen Zustands**
- Eine Menge von definierten **Aktionen**
- Ein Menge von **Zielbeschreibungen**
  - Inkl. **Zielzustand**

Repräsentation des Plans

Azyklischer, gerichteter Graph

DAG (*engl.* **d**irected **a**cyclic **g**raph)

Bestehen aus Zuständen, Übergängen

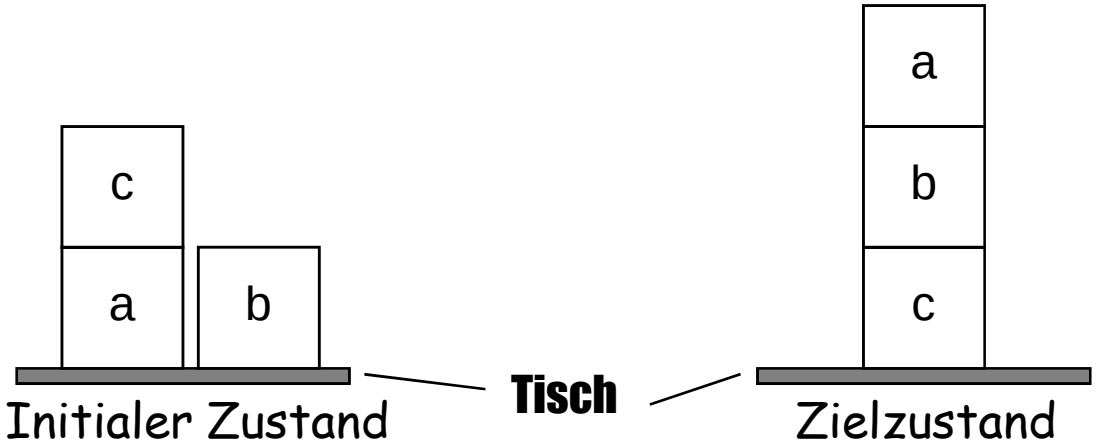
Keine Schleifen

Voraussetzung

Die Welt, in der Pläne erfolgreich realisiert werden können, muss **größtenteils vorhersehbar sein**, wenn nicht **vollständig deterministisch**.

Ist deterministisch, wenn zu jedem Zeitpunkt genau definiert ist, wie weiter vorgegangen werden soll.

Mikrowelt-Beispiel: Beschreibung von Start- und Zielzustand



Beschreibungselemente

```
clear x          auf x ist nichts
holding x        der Agent hält x
on x y           x befindet sich direkt auf y
ontable x        x befindet sich direkt auf dem Tisch
...
```

Die Aktionen / Aktionsschemata

Aktionsschema (generisch)

```
(    < Aktionsname      >   < Variablenliste >
     < Vorbedingungen   >
➔    < Nachbedingungen  >   )
```

Die Aktionen / Aktionsschemata: Beispiele

```
(       < Aktionsname     >    < Variablenliste >
        < Vorbedingungen  >
→       < Nachbedingungen >    )
```

```
(PICKUP (block))
     ((ONTABLE block)
      (CLEAR block))
→
((HOLDING block)
 (NOT (CLEAR block))
 (NOT (ONTABLE block))))
```

```
(PUTDOWN (block))
     ((Holding block)
→
((CLEAR block)
 (NOT (CLEAR block))
 (NOT (ONTABLE block))))
```

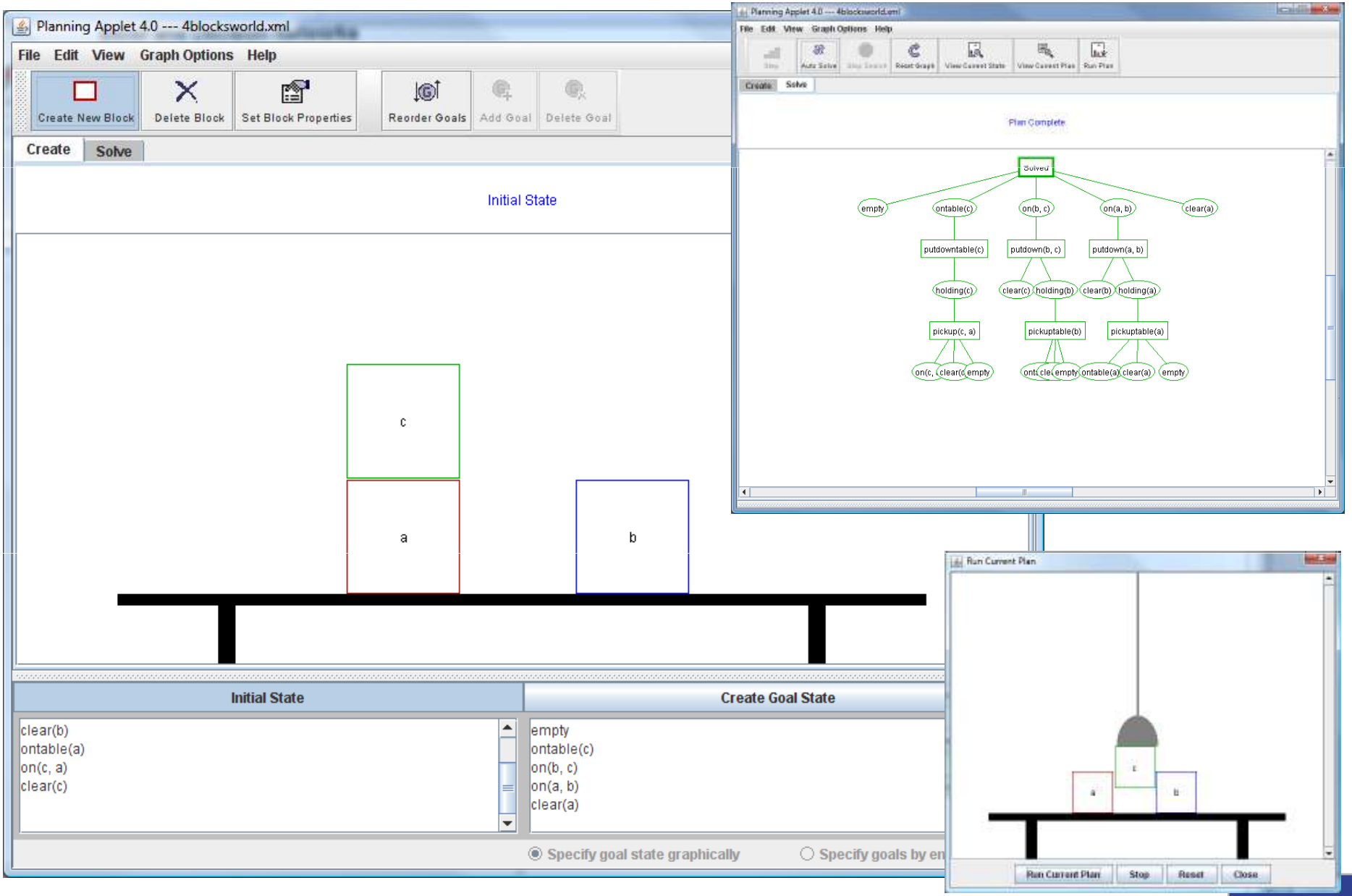
Nachbedingung steht (i.d.R.) **im Widerspruch zur Vorbedingung**

STRIPS

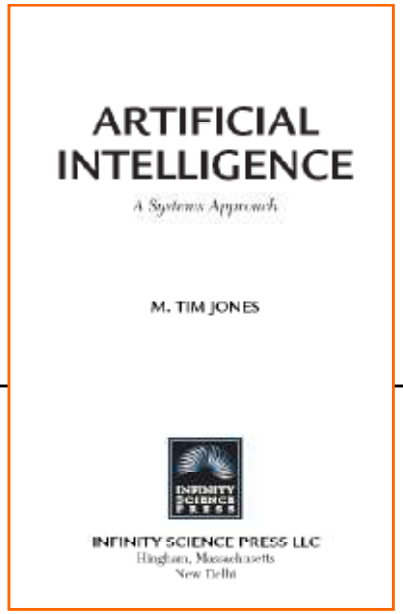
- **S**tanford **R**esearch **I**nstitute **P**roblem **S**olver (Fikes, Nilsson, 1970‘er)
- Zur Planung von Roboter-Agenten, um in einer Blockwelt navigieren zu können
  - Ausweitbar auf andere Gebiete mit Planungsproblematik
- PL1, um die Welt zu beschreiben

$$(\forall o \forall x \forall y)\big((AT(o,x) \wedge (x \neq y)) \Rightarrow \neg AT(o,y)\big)$$

Demo:
http://www.aispace.org/downloads.shtml



- Roboter (Planungprozess)
  - Robotik, STRIPS
- **Agenten (in Computerspielen)**
  - **FSM, Agenten-Architektur, PEAS**
  - ***Utility function***



Applications of AI Algorithms in Video Games

- **Video game environments** (such as can be found in realtime strategy games or first-person-shooters) provide a useful **test-bed for the application and visualization of AI techniques**.
- Games themselves have become a **huge industry** (it's **estimated that games gross more than movies**), so the development of AI techniques with the associated constraints that can be found in games (such as minimal CPU allotment) can be very beneficial.
- It's also possible to allow different **algorithms and techniques to compete against one another** in these environments **to understand their subtle differences and advantages**.

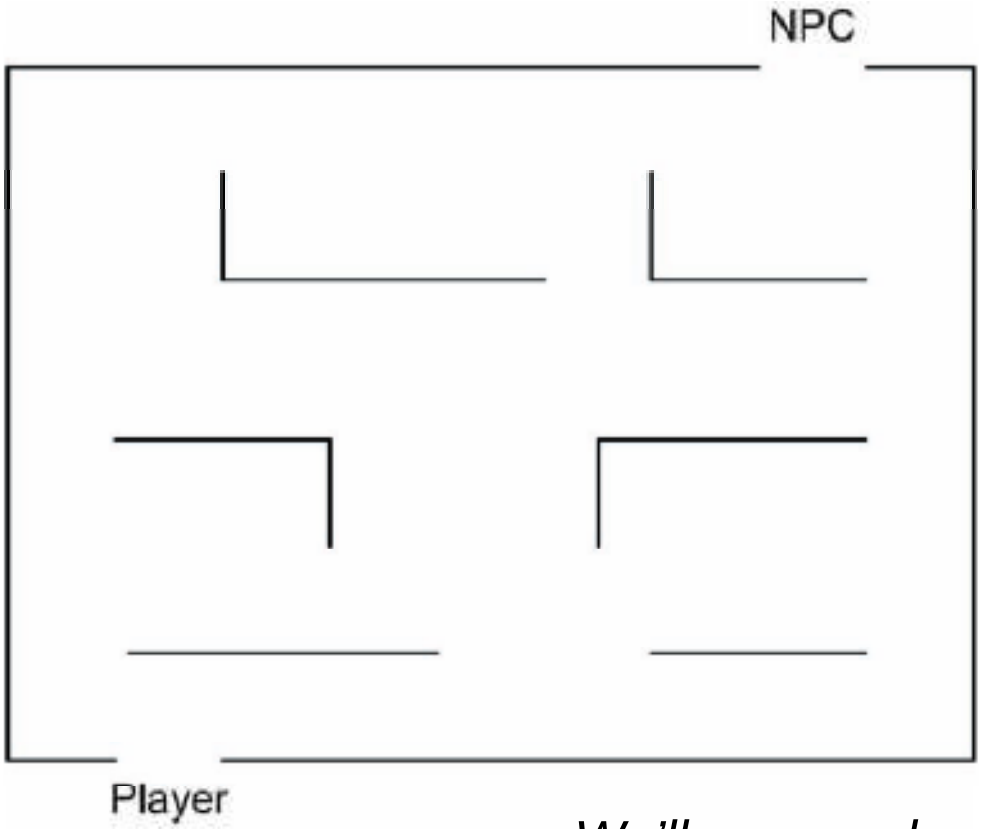
Applications of AI Algorithms in Video Game

- In addition to the entertainment value of video games, the techniques for **building believable characters** also finds value (and research funding) in military applications.
- For example**, flight combat simulators** that **mimic the strategy and tactics of veteran pilots**, or the **hierarchical and disciplined behavior of troops** on the ground in **battle management simulators**.
- Each of these applications requires intelligent algorithms that may differ in embodiment, but evolve from the same set of techniques.

Movement and Path-finding

- The object of path-finding in many games from first or third-person-shooters, to real-time strategy games is **identifying a path on a map from point A to point B**.
- Any **map can be reduced to a graph**, where the **nodes** are the **places** that can be visited, and the **edges** are the **paths between the nodes**.
- Reducing a map in this way does a couple of things.
  - First, it potentially **reduces a map with an infinite number** of points into a **graph with fewer points**. The edges, or the paths between nodes in the graph, define the various ways that we can travel around our graph (and our map).

Movement and Path-finding



We'll assume here, for the sake of simplicity, that the NPC always sees the player. In a more complicated system, the player would need to be in the NPC's field-of-view (FOV) in order for the NPC to identify the player's presence.

Simple graph for our NPC

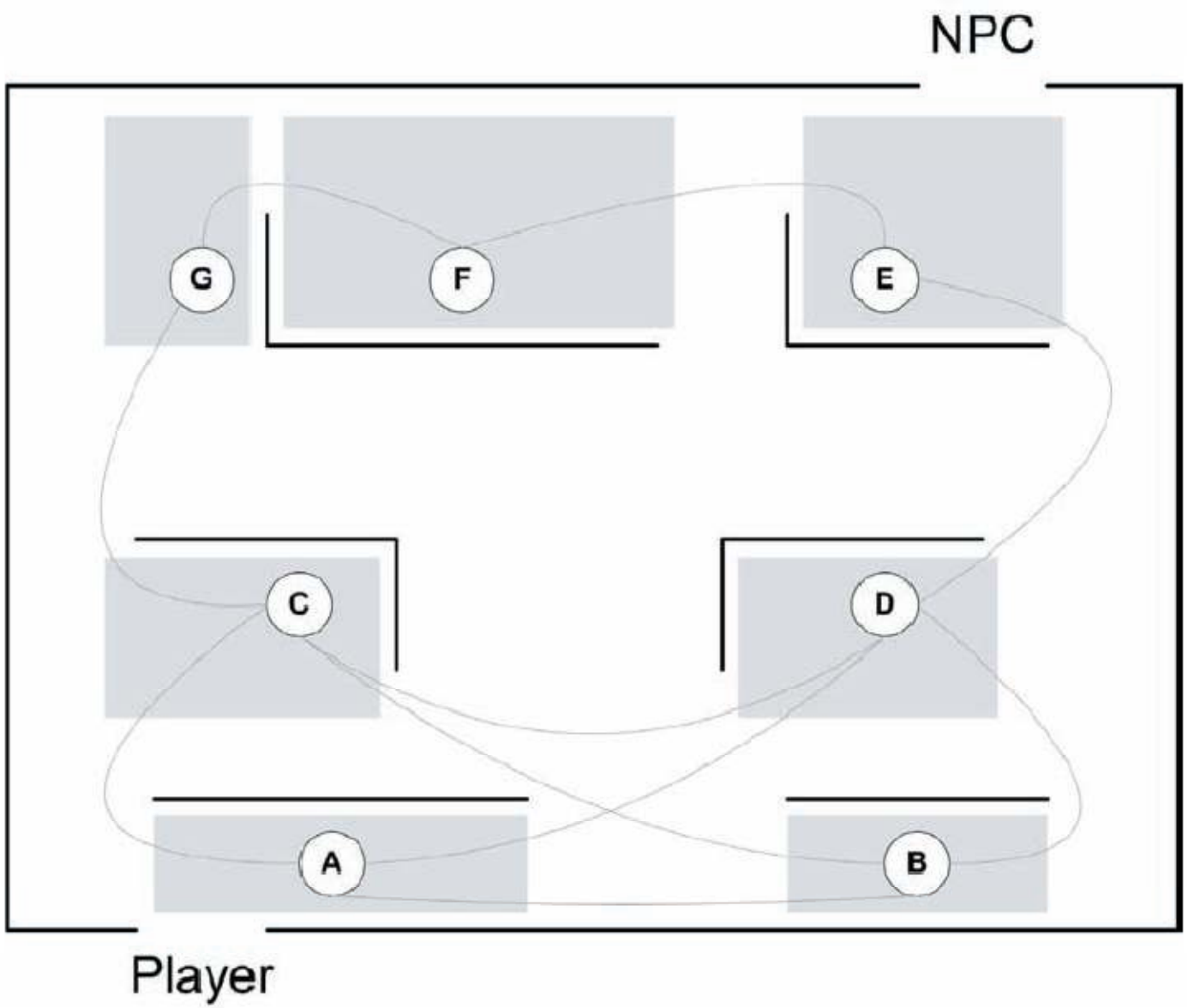


Table Lookup with Offensive and Defensive Strategy (**Offensiv**)

Player (Opponent)

NPC	A	B	C	D	E	F	G
A		B	C	D	D	C	C
B	A		C	D	D	C	C
C	A	B		D	D	G	G
D	A	B	C		E	E	C
E	D	D	F	D		F	F
F	G	E	G	E	E		G
G	C	C	C	C	F	F	

Offensive Strategy

For example, if the **NPC is at node E**, and the **player is around node A**, **then the NPC will** use the offensive strategy table and **move to node D**. If the **player then moves from node A to node C**, we use the table again for the **NPC at node D**, and the **player at node C**, which **results in the NPC moving to node C** (on the attack).

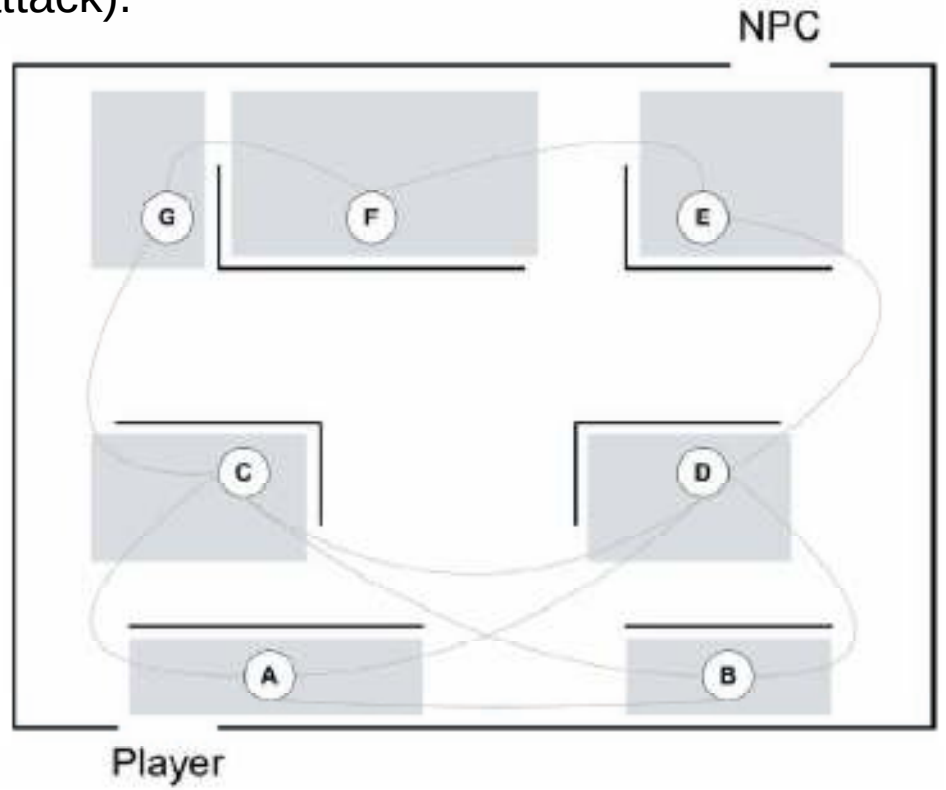


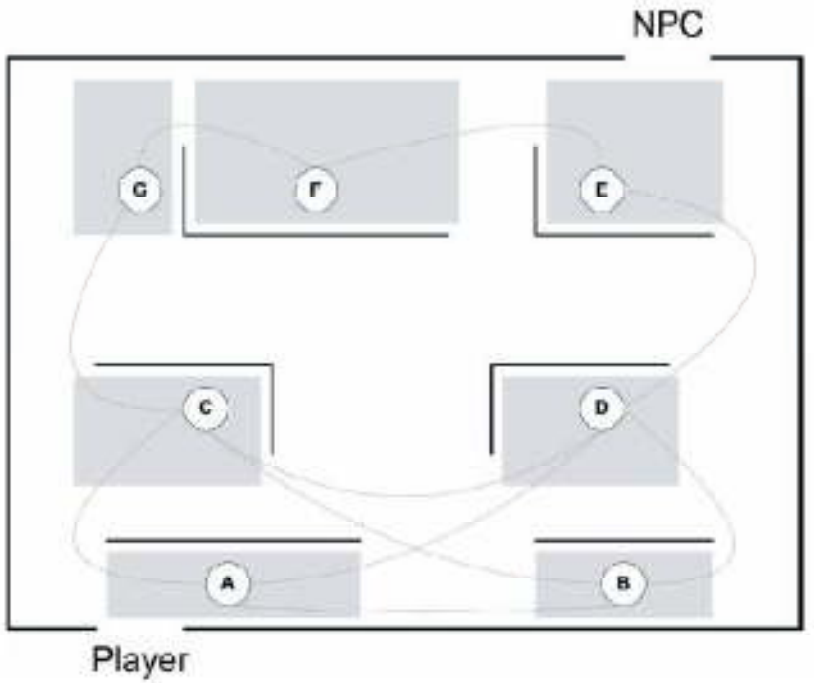
Table Lookup with Offensive and Defensive Strategy (**Defensiv**)

Player (Opponent)

NPC	A	B	C	D	E	F	G
A		C	D	C	-	B	B
B	D		G	C	A	-	-
C	D	G		G	A	B	D
D	E	E	E		A	B	B
E	-	F	-	F		D	D
F	E	G	E	G	G		E
G	F	-	F	-	C	C	

Defensive Strategy

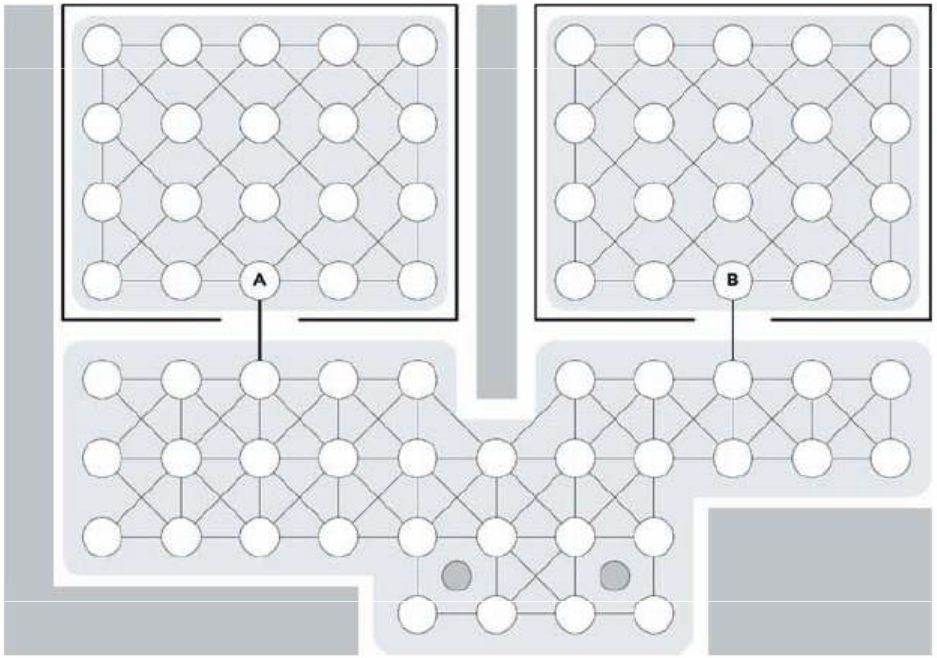
Take, for example, the **NPC at node D** and the **player again around node A**. The lookup table returns a **move from node D to node E**, essentially putting distance between us and the player. If the **player then moved to node D**, the lookup table would return **node F**, moving away from an eventual move by the player to node E. Note that **in some cases, the best move is to not move at all**. If the NPC was at node G and the player at node B, the return value from the table is '-' indicating to simply stay put.



Ob offensive oder defensive Strategie, entscheidet die Gesundheit des NPC.

Multiple connected maps

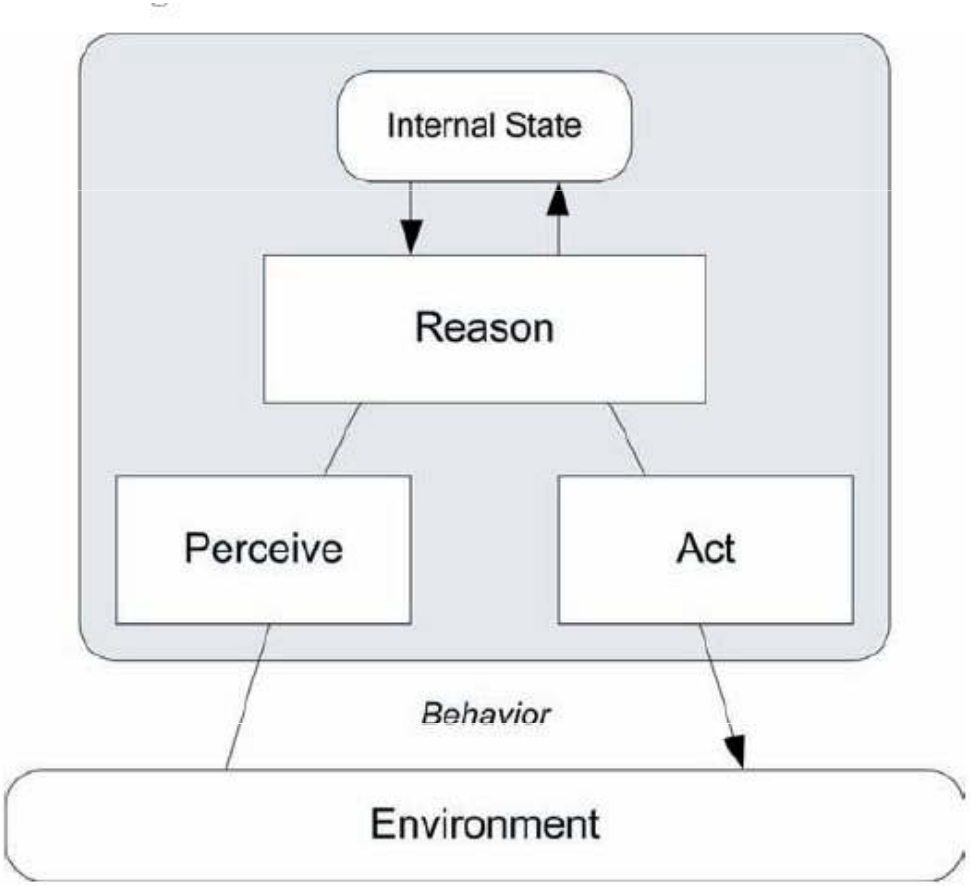
- In large environments, it can also be possible to **segregate a map into multiple connected maps**, each having funnel points for which an NPC may travel.
- The Figure shows a map of three zones. Rather than including a single lookup table for all points, **three separate lookup tables** would exist.



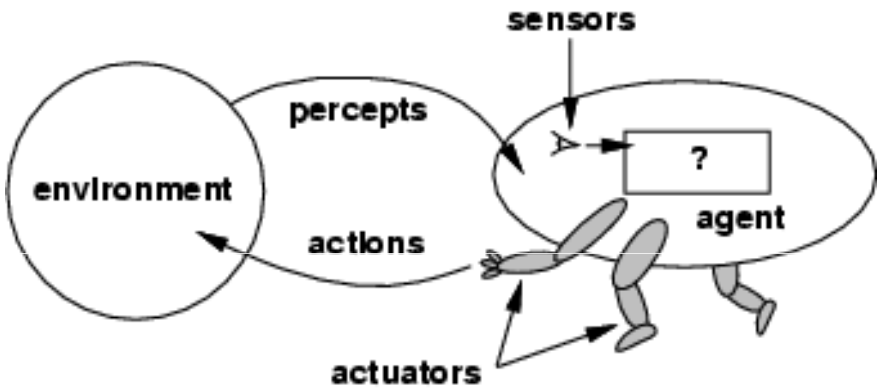
NPC Verhalten

- The behavior of an NPC can't be considered in isolation, because **behavior is ultimately grounded in the environment**.
- The NPC must be able to ***perceive*** its environment (see, hear, etc.).
- With the information perceived from the environment (as well as internal state information such as motivation), the NPC can ***reason*** about what should be done. The result of **reasoning** is potentially an **intentional act**, which is performed as an ***action***.
- This **action** (and subsequent actions) is what we externally view as **the agent's behavior**.

NPC Verhalten und Architektur



»Ein Agent ist alles, was seine Umgebung über **Sensoren** wahrnehmen kann und in dieser Umgebung durch **Aktuatoren** handelt.«



Ein Agent handelt rational. D.h., wenn er weiß, dass seine folgende Aktion effektiv zur Problemlösung beiträgt, wird er diese Aktion durchführen.

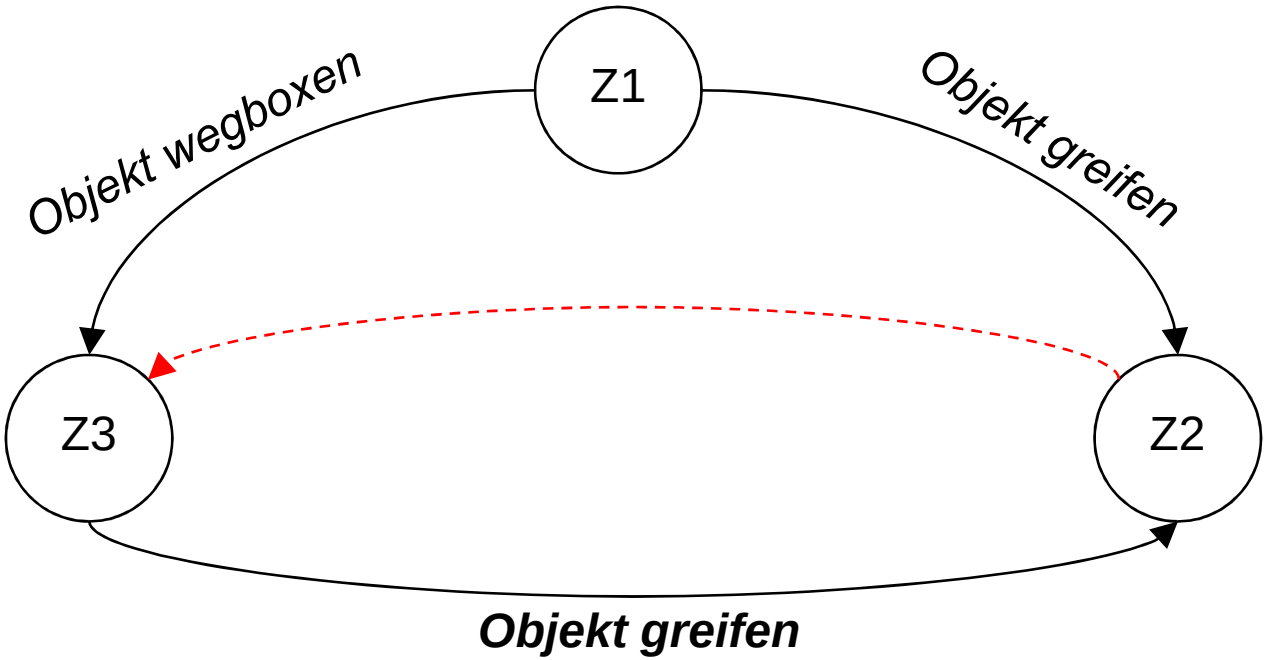
Aktionen via *Finite State Machines*

- One of the **simplest methods**, and also one of the **most common**, is the state machine.
- State machines consist of a **set of states**, and **arcs**, between the states that **define the conditions necessary for transition**.

DAG Beispiel: Roboterarm greift oder drückt ein Objekt weg

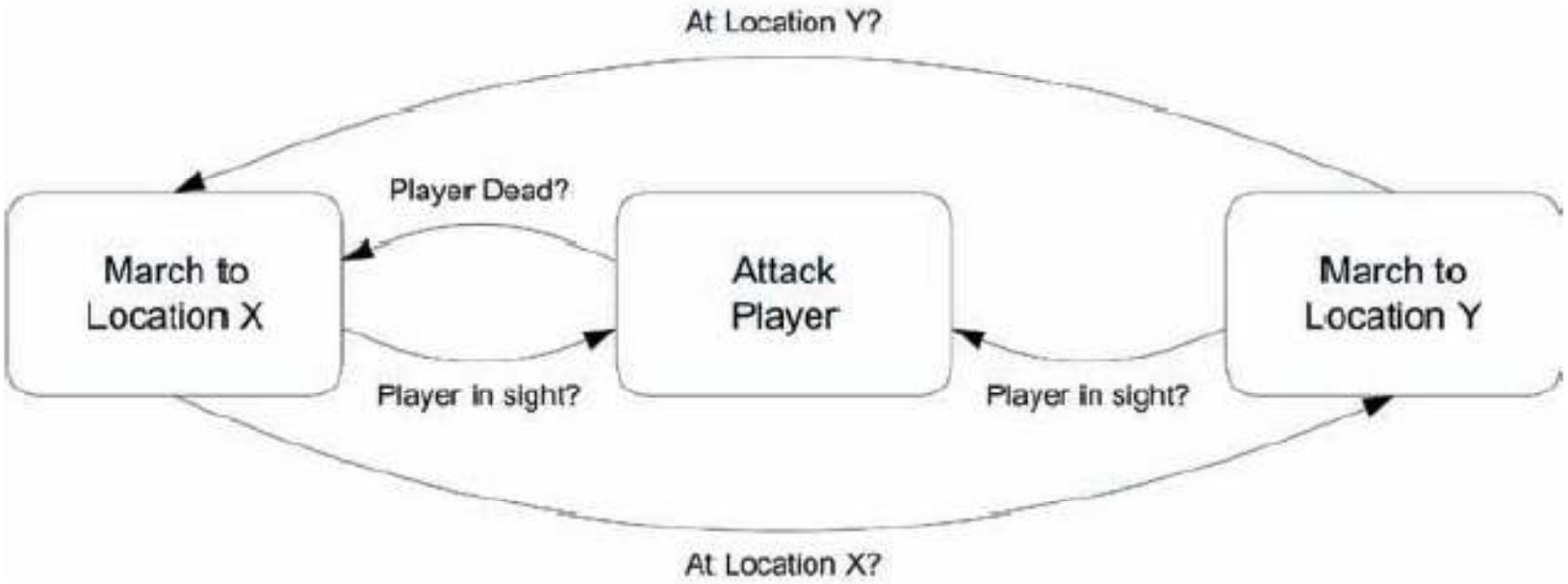
Aktionen

Zustand 1 (Z1):	*Objekt auf Tisch*
Zustand 2 (Z2):	*Objekt im Greifarm*
Zustand 3 (Z3):	*Objekt auf dem Boden*
Kante von 1 zu 2:	*Objekt greifen*
Kante von 2 zu 3:	*Objekt wegboxen*
…	

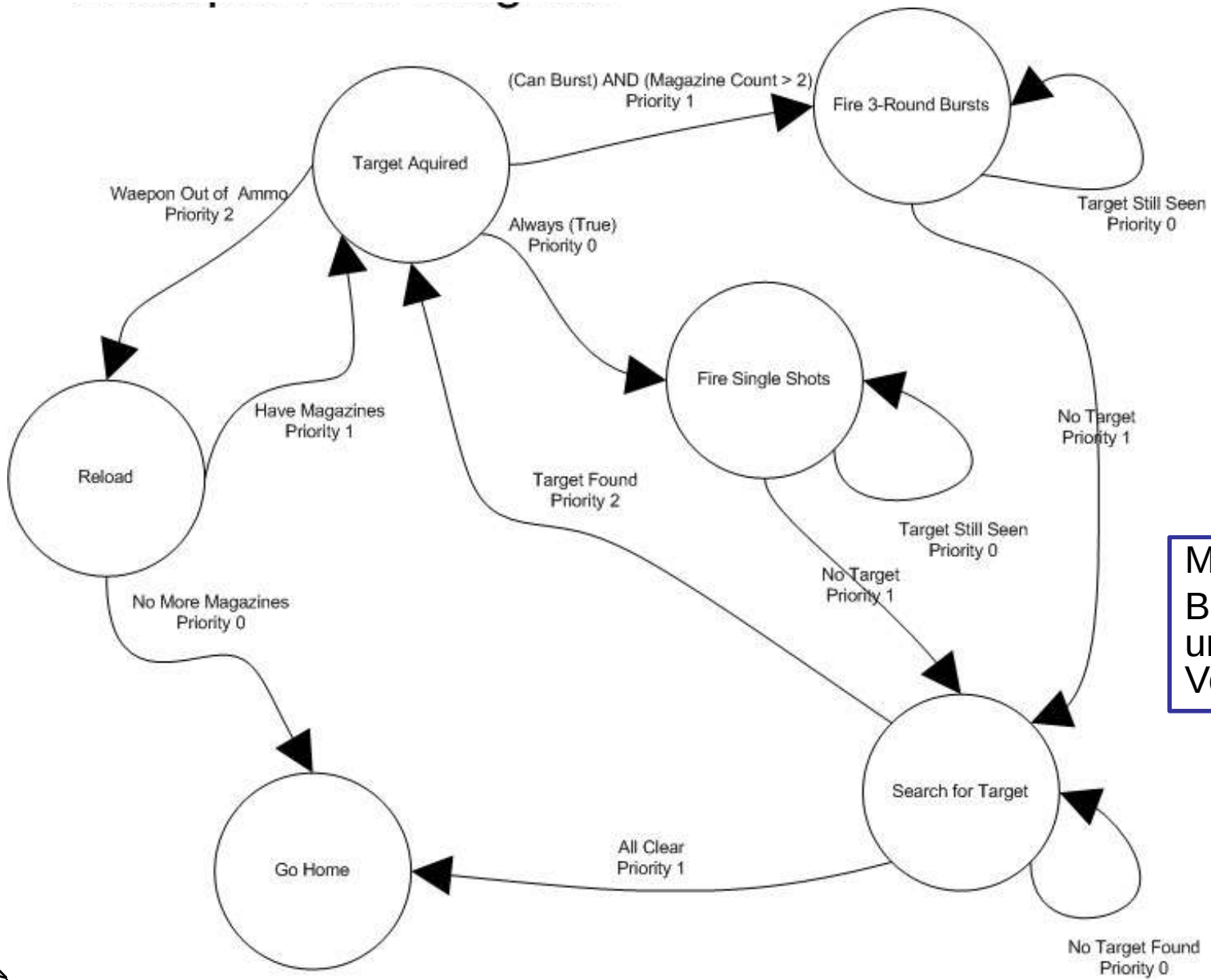


Finite State Machine fürs Herumwandern und Angreifen

Our NPC sentry defined by the [finite] state machine has two basic functions in life. The NPC marches between two locations, guarding some entry from the player. When the player is in sight, the NPC fights to the death. The state machine implements this as three simple states (one can think of them as *mental states*). In the first state, the NPC marches to the location identified as X. It continues to march to X unless one of two things happen. If the NPC reaches location X, it's at its destination for the state, and we transition to the alternate march state. If the NPC sees the player, it attacks. If neither of these events occurs, the NPC continues marching.When the NPC sees the player, by entering its field-of-view, the FSM transitions to the attack state. In this state, the NPC fights to the death. If the NPC dies, then the state machine is no longer. If the NPC defeats the player, it begins marching again toward location X. There's not much to the FSM, but they are simple and easy to debug. They are also very predictable, but it's possible to add transition probabilities to give the NPC a small element of randomness.



Komplexeres FSM mit Prioritäten bei mehreren Möglichkeiten



Maschinelles Lernen:

Beobachte zwei Spieler und lerne/imitiere ihr Verhalten.

Layered Behavior Architectures

At level one, our NPC performs his guard duty, dutifully marching along his route.

If the player comes into view, the NPC switches to the attack layer to rid the environment of the pesky player. From **Brookes' architecture**, the ***Attack* layer subsumed** (**took priority over**) the *March* layer.

If no players are in the field-of-view, and the NPC needs to be healed or replenish his ammunition, the *sustain* layer takes over to perform those actions.

Finally, the NPC will turn to the *survive* layer if more than one player is seen in the fieldof-view. As the constraints are met within the layers, the NPC will default to the lowest layer.

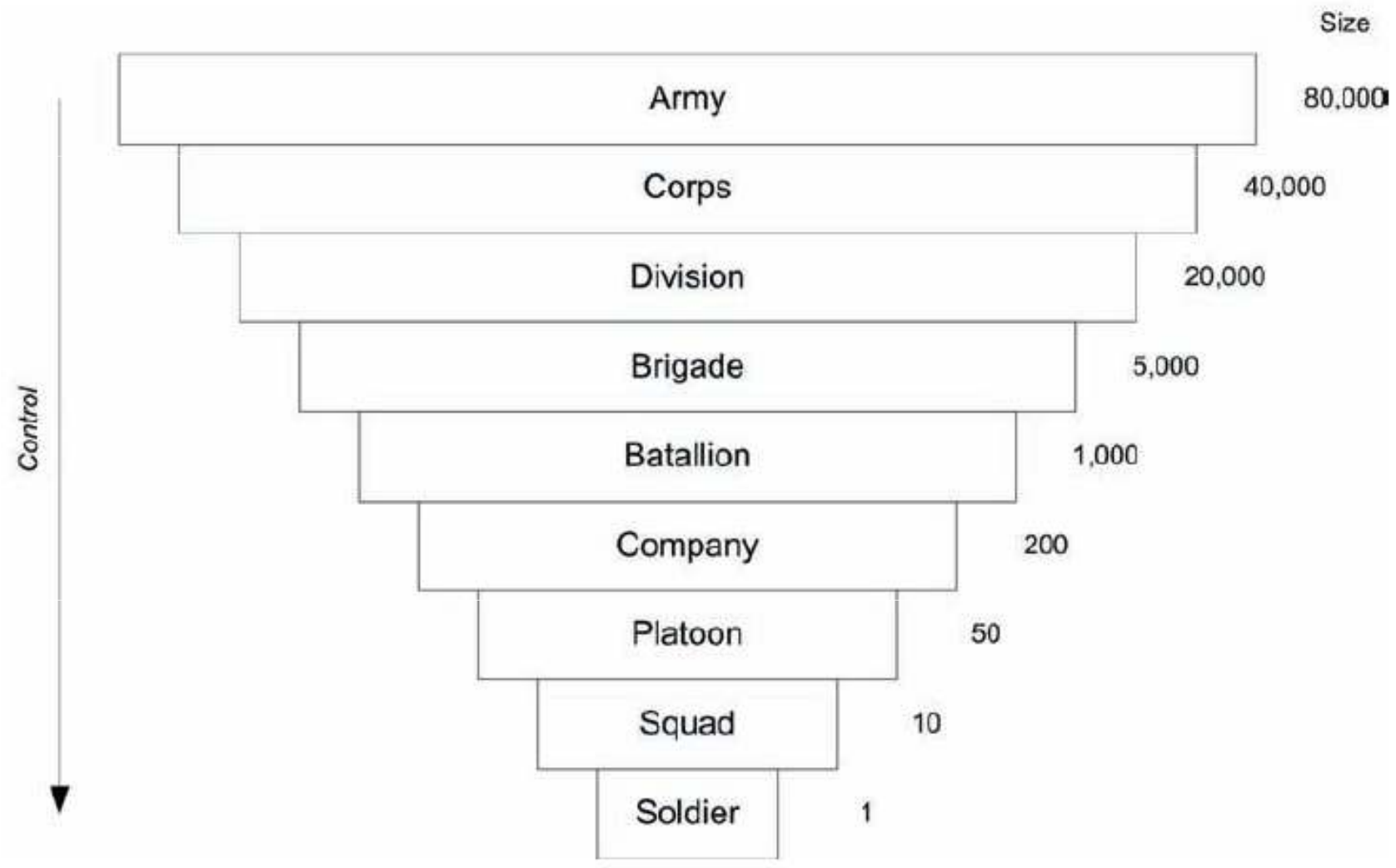
Therefore, once the player is disposed of, and the ammo and health are returned, the NPC will return to the march in level one.

Within each of these layers, individual FSMs can be used to implement the relevant behavior. In this way, entirely new state machines are consulted based on the current behavior layer for the NPC.

Team AI

- In many games, there's not just a single NPC soldier that we're fighting against, but an entire army that must work together in harmony with a single or handful of goals in mind.
- The control can exist at a **number of levels within a hierarchy**, from the squad leader directing troops to flank an enemy in the field using real-time battlefield information, to the general managing his entire army to implement the higher-level military strategies.
- Managing an overall army can entail a large number of problems, from scheduling and production, to resource allocation, to the overall strategy and tactics of the force in the field.
- The problem here can be simplified greatly by minimizing the number of levels or organization. First, the individual soldiers simply follow orders,and unless they are routed, can fight to the death. These individual units can be programmed using finite state machines, or other simple mechanisms for behavior.
  - *Following orders is a requirement, but there's also something to be said for autonomy at the soldier level, taking advantage of a local situation to improve the chances of a global win.*

Team AI: Strukturierung in Globale Strategie, Taktik, ausführende Einheiten



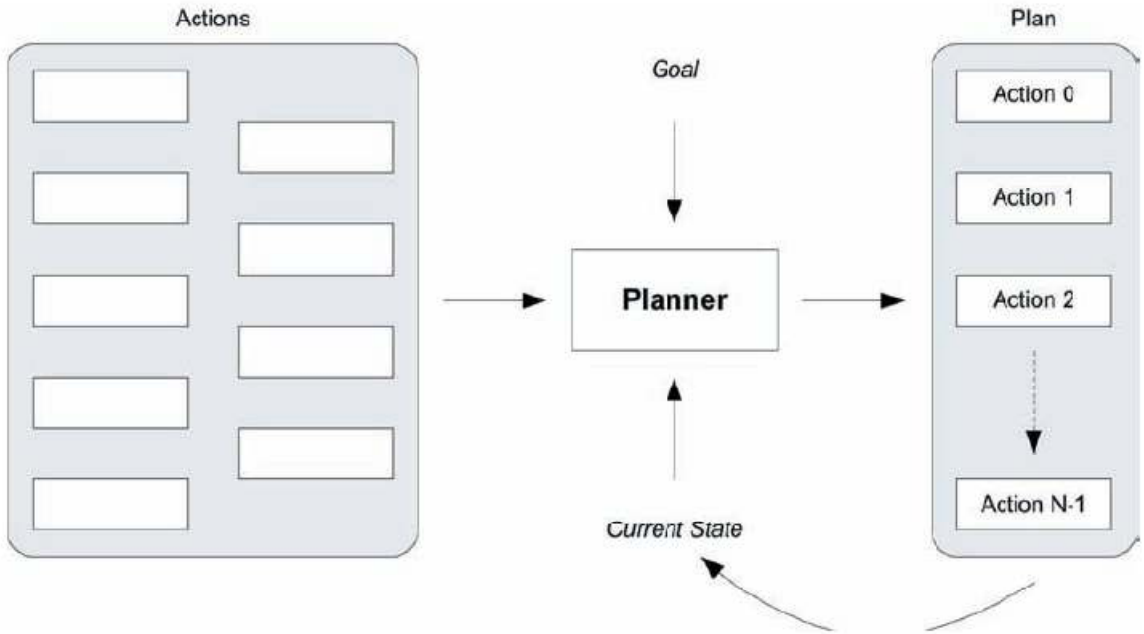
Goals and Plans

- An interesting mechanism for high-level **control** over a hierarchy of military units is i**n defining a goal**, and **then creating a plan** to **reach that goal**.
- Also necessary is the need to **reformulate a plan**, when it eventually fails.

Goals

- First, there's the ***goal***.
- A goal can be an **end-goal** (for the end of the game) or an **intermediate goal** that moves us closer to an end-game situation.
- Formally, a **goal is a condition that is desired to be satisfied**.
  - Example goals include taking an enemy position, destroying a bridge, flanking an enemy, etc.

Plan, *Planner*



- **To reach a goal, we must perform a series of *actions***, which are **independent steps** within a plan. Each action potentially has a **prerequisite that must be met in order for the action to be performed**.
- Example actions include moving a set of NPC units to a location, attacking the enemy, planting a charge, etc. The ***plan,*** then, **is a set of actions that when performed in the given order**, **achieve a goal.** The method by which a plan is formulated is through the use of a *planner*.

Beispiel: Geplanter Angriff

- As an example, let's say we have **two NPCs that desire to attack a player** that is firing from a covered position.
- An **effective attack** is for one NPC to **provide covering fire at the player**, while the **other NPC moves to a flanking position** to be in a better position to attack the player.

Formulierung der Regel: Ziel, Vorbedingung, Plan

Goal: Eliminate(Player)

Prerequisites:

Covered_Position(Player)

Alive(Player)

Plan:

Action-1: Identify_Flanking_Position(NPC_1)

Action-2: Covering_Fire(NPC_2)

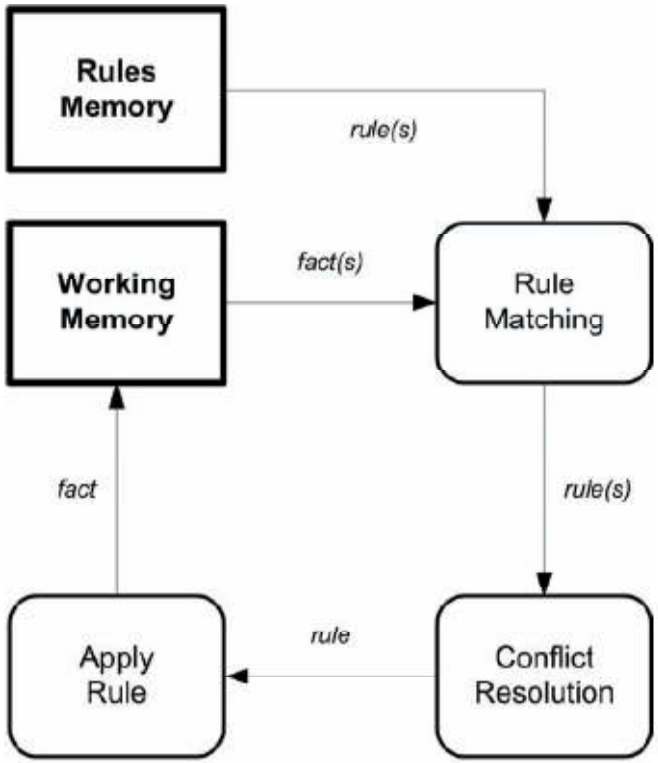
Action-3: Move_to_Flanking_Position(NPC_1)

Action-4: Fire_at_Player(NPC_1)

Action-5: Rush_Player_Position(NPC_2)

Rule-Based Programming zur Verhaltenssteuerung

- A rule-based system is made up of **two memories**, one that holds facts, and another that holds a set of rules that exist to determine the behavior.
- A **rule-matching algorithm is applied to the facts**, and those rules that match facts are saved into the conflict-set.
- This set of rules is then reviewed and one rule picked to fire in a process called **conflict resolution**.
- The **rule is** then **applied** and the **working memory** is **updated**.



Beispiel

- Consider the following four facts in our simple knowledge base:

```
(opponent-1 size large)
(opponent-2 size small)
(army size large)
```

Regeln

- Rules are made up of two things, **antecedents** and **consequents**.
- If you think about a rule as an if/then construct, then the ‘if’ is the antecedent, and the ‘then’ is the consequent.
- Consider the following simple comparison:

```
if (    (opponent.size <= SMALL_SIZE) &&
        (army.size >= LARGE_SIZE)) {
    attack_opponent();
  }
```

A simple rule that encodes this behavior could be implemented as follows

```
( rule “Attack Opponent Based Upon Size”
     (?opponent size small)
     (army size large)
  ==>
  (army attack ?opponent) )
```

Once this rule fires, our working memory will exist as:

```
(opponent-1 size large)
(opponent-2 size small)
(army size large)
(army attack opponent-1)
```

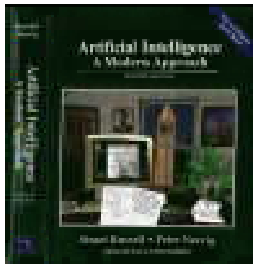
Weitere essentielle NPC (und Agenten-) Kriterien

- **Lernfähigkeit**
  - Soll Agenten erlauben, in zunächst unbekannten Umgebungen zu arbeiten und darin kompetenter zu werden, als das Ausgangswissen es eigentlich erlauben würde
- **Autonomie**
  - Fähigkeit zu Handeln und Entscheidungen zu fällen, unabhängig vom Programmierer oder Auftraggeber (User)
  - Bspw.: Shopping-Agent
    - Einkauf von Gütern auf Basis von Benutzerbedürfnissen
    - Entscheidungen, was zu Kaufen ist, muss ohne »Rücksprache« mit dem Benutzer möglich sein.
  - Autonomie ist ein wesentlicher Unterschied zu vielen anderen AI-Techniken/-Technologien

Second Life

Die Umwelt/Die Umgebungen

- Problemfelder, innerhalb derer Agenten zu arbeiten haben
- **»PEAS«** als allg. Beschreibungsform
  - *Performance* (Leistung)
  - *Environment* (Umgebung)
  - *Actuators* (Aktuatoren/Effektoren)
  - *Sensoren* (Sensoren)



PEAS: Taxifahrer-Agent

Performance measure

- safe, fast, legal, comfortable, maximize profits

Environment

- roads, other traffic, pedestrians, customers

Actuators

- steering, accelerator, brake, signal, horn

Sensors

- cameras, sonar, speedometer, GPS

PEAS für NPC-Agentenumbegung

Performance measure

- hat mehr Punkte als Spieler, kann heilen, nachladen, …

Environment

- Map (Gebäude, Wald, …)

Actuators

- Beine, Waffe, …

Sensors

- Sicht, Gehör, (Allwissenheit), …

Nutzenbasierte (NB-) Agenten

- Zusätzlich zum Erreichen eines Ziels versucht ein Nutzenbasierter Agent einen **Nutzen-Wert** zu maximieren.
  - »A **utility-based agent** is similar to a goal-based agent, but in addition to attempting to achieve a set of goals, the utility-based agent is also trying to maximize some **utility value**. The **utility value** can be thought of as the happiness of the agent, or **how successful it is** being.«
- Der »happiness«-Wert muss errechnet werden: Funktion.
- Rationale Agent besitzen im Idealfall stets Nutzen-Funktionen (zur verbesserten Suche, Planung, …)

Utility function

- In other words, given a particular **state of the world**, an agent is able to use its **utility function to derive a score**, or **utility value**, that **tells it how "happy" it is** in that state or […]«
  - Gegeben: Weltzustände
  - Funktion: Berechnet Wert auf Basis der Zustände
  - Wert: Bestimmt den »happiness«-Faktor

Nutzen-Funktion: Beispiel

Von den Psychologen Tversky, Kahnemann durchgeführtes Experiment (1982)

1. A or B:

   $A = 80\%$ chance of winning \$4000

   $B = 100\%$ chance of winning \$3000

Wofür entscheiden Sie sich?

Nutzen-Funktion: Beispiel

Von den Psychologen Tversky, Kahnemann durchgeführtes Experiment (1982)

2. C or D:

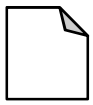
$C = 20\%$ chance of winning \$4000

$D = 25\%$ chance of winning \$3000

Wofür entscheiden Sie sich?

»Most subjects choose **A**, **rather than B**; and **C**, **rather than D**.«

Die Versuchspersonen haben rational gehandelt: wie ist das messbar?



Utility function für einen Casino-Agenten

A = 80% chance of winning \$4000 **`E(A) = 0.8 * 4000 = 3200`**

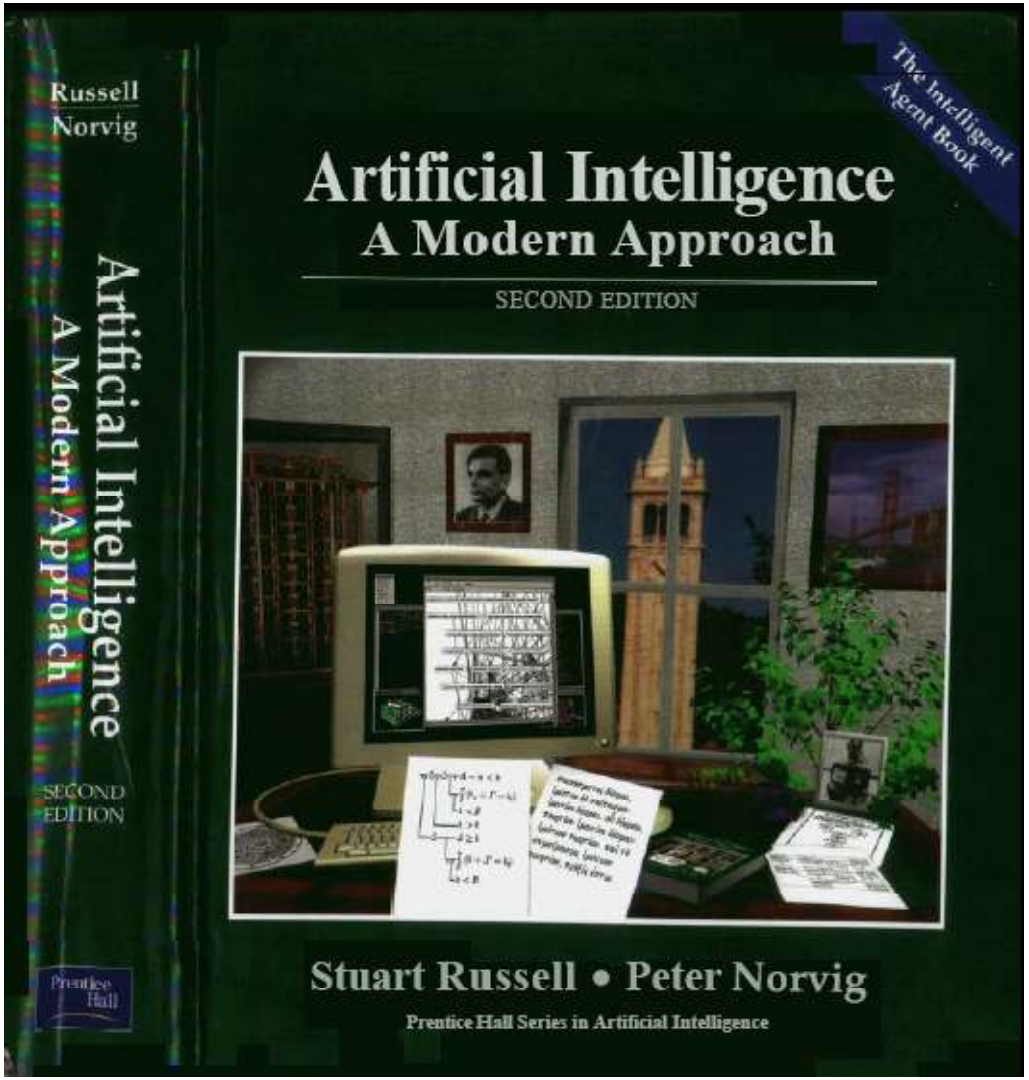
B = 100% chance of winning \$3000 **`E(B) = 1.0 * 3000 = 3000`**

C = 20% chance of winning \$4000 **`E(C) = 0.2 * 4000 = 800`**

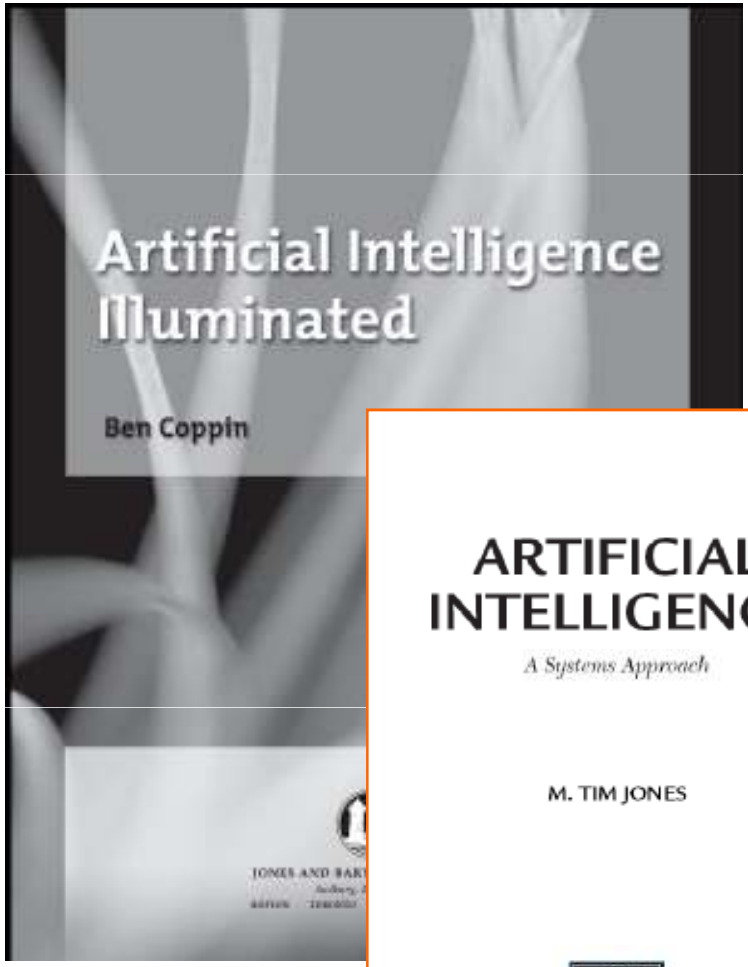
D = 25% chance of winning \$3000 **`E(D) = 0.25 * 3000 = 750`**

Rational sind Möglichkeiten *A, C* auf Basis ihrer höheren *utility values.* (Der Agent wäre quasi unglücklicher mit den anderen Möglichkeiten.)

Referenzen



Kapitel 2



Kapitel 19

Kapitel 4