

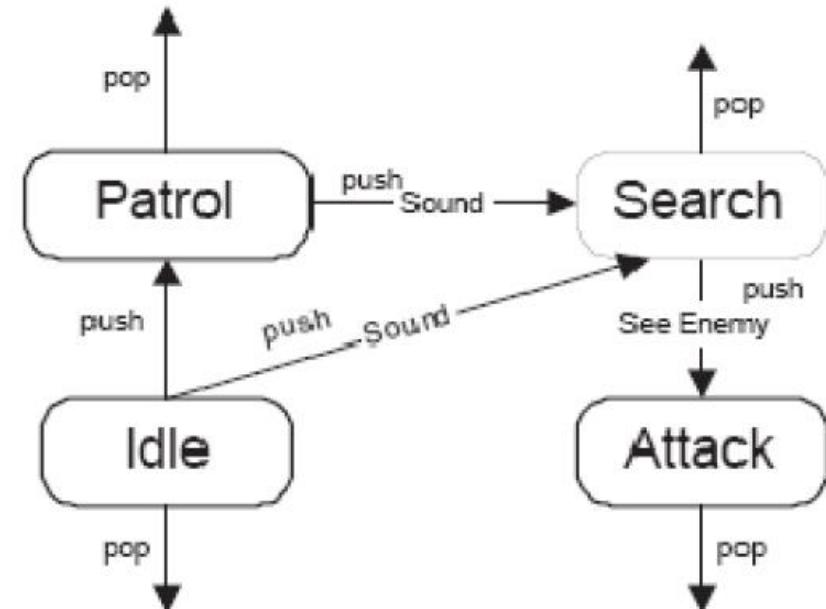


Einführung in die Spieleprogrammierung

Künstliche Intelligenz (KI)

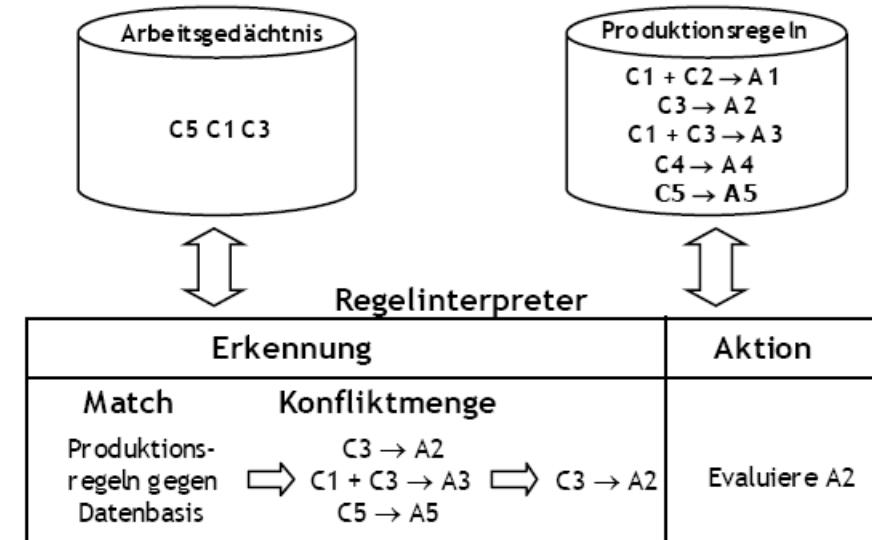


- Keller: Speichern vorheriger Zustände
 - Gedächtnis, Rückkehr zu vorherigen Verhaltensweisen
- Zustands-Stack
- Transitionen:
 - push
 - pop
 - replace
- Beispiel: Wächter in Thief 3



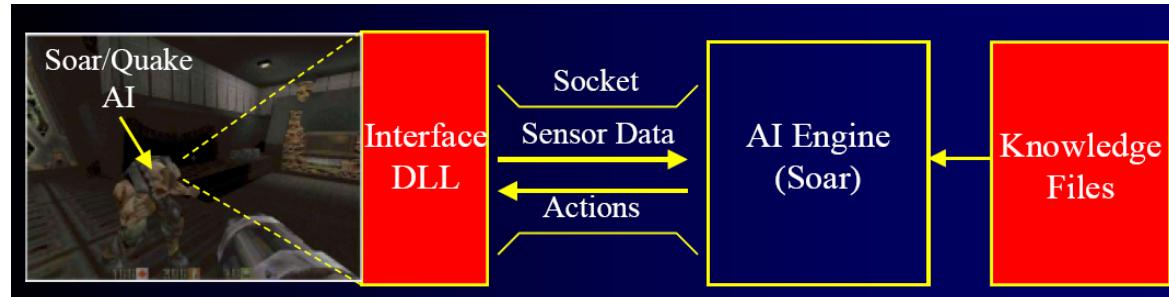
- Jede Produktion (oder Inferenzregel) $A \rightarrow B$ besteht aus zwei Teilen:
 - Prämisse & Konklusion
 - Antezedenz & Konsequenz
 - Bedingung & Aktion
- Die Produktionen werden über der Datenbasis ausgewertet, die auch als Arbeitsspeicher (Working Memory) bezeichnet wird
- Konfliktauflösung, wenn mehrere Regeln feuern:
 - Reihenfolge: Erste Regel, die zutrifft
 - Spezifität: Spezifischste Regel
 - Priorität: Regeln haben Prioritäten, höchste wird gewählt

Architektur eines Produktionsregelsystems



Beispiel: Soar / Games Projekt (van Lent & Laird, University of Michigan)

- Interface zwischen auf Soar (State, Operator and Result) basierender KI-Engine und unterschiedlichen Spiele-Engines:
Soar / Quake II, Soar / Descent 3, Soar / Unreal, Soar / Half-Life



```
If enemy visible and  
my health is > very-low-health-value (20%) and  
his weapon is not much better than mine  
THEN propose attack
```

```
If enemy visible and  
my health is < very-low-health-value (20%) or  
his weapon is much better than mine  
THEN propose retreat
```

```
If no enemy visible or recorded and explored level  
THEN propose wander
```

```
IF I see a weapon that is much better than any I have  
THEN I need that weapon
```

```
IF my health is less than low-health-value (40%)  
THEN I need any health object
```

```
IF two get-item operators are proposed  
THEN prefer selecting the one for the closer object
```



Bewertung von regelbasierten Systemen

Vorteile:

- Intuitiv
- Ausdrucksstark
- Deklarativ
- Einfach zu implementieren und zu erweitern
- Durch Fuzzy-Logik erweiterbar

Nachteile:

- Speicherintensiv
- Hoher Rechenaufwand
- Evtl. schwer zu debuggen

Motivation (aus: Buckland - Programming Game AI by Example)

Menschen kommunizieren über vage, unscharfe („fuzzy“) Begriffe, z.B.
Rezept in Kochshow:

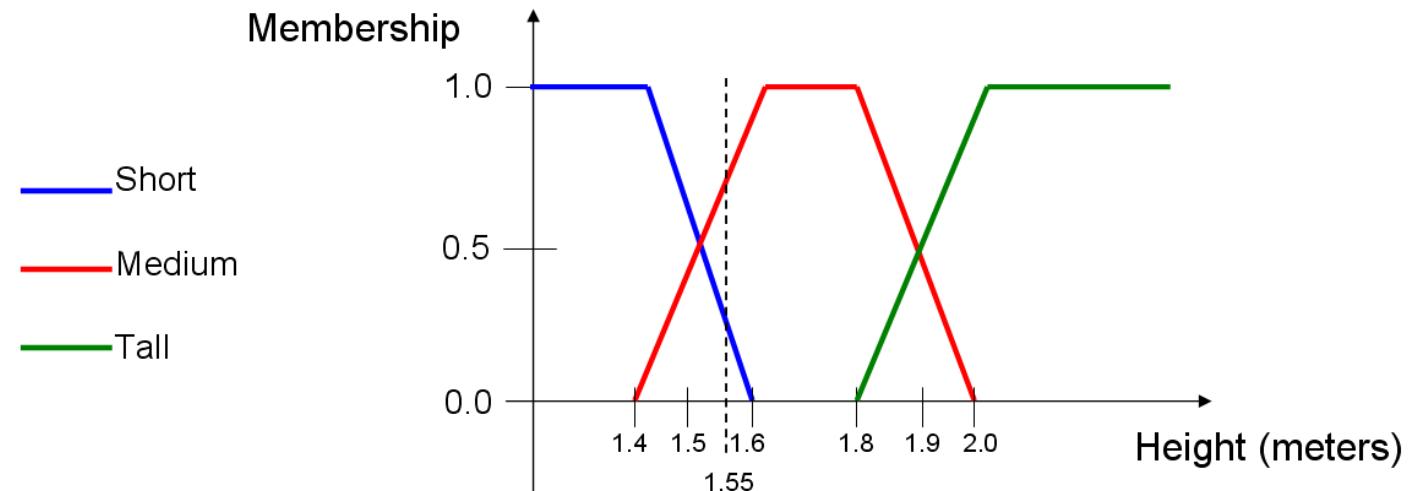
1. Cut two slices of bread **medium thick**.
2. Turn the heat on the gridle on **high**.
3. Grill the slices on one side until **golden brown**.
4. Turn the slices over and add a **generous helping** of cheese.
5. Replace and grill until the top of the cheese is **slightly brown**.
6. Remove and sprinkle on a **small amount** of black pepper.

Wie kann man diese Begriffe einer KI beibringen? Was genau ist
medium thick? 1 cm? Was sind dann 9 mm?

- Geschichte
 - Basiert auf Fuzzy Mengenlehre
 - Eingeführt 1965 von Lotfi Zadeh (Berkeley University)
- Fuzzy Mengenlehre vs. herkömmliche Mengenlehre (“crisp”):
 - Mengenlehre: ein Element gehört entweder zu einer Menge oder nicht
 - Fuzzy Mengenlehre: ein Element kann teilweise zu einer Menge gehören
 - In der Mengenlehre wird i.d.R. ein Schwellwert als Kriterium verwendet, z.B. sind alle Personen größer 1.8m in der Menge “tall”, alle gleich oder kleiner 1.8m in der Menge “small”
 - In der Fuzzy Mengenlehre gehört z.B. eine Person, die 1.85m groß ist zu 100% zur Menge “tall”, eine Person mit 1.72m nur zu 20%
- Fuzzy Logik lehnt sich oft an das reale Leben und die menschliche Denkweise an
- Fuzzy Logik kann in Regelbasierten Systemen verwendet werden

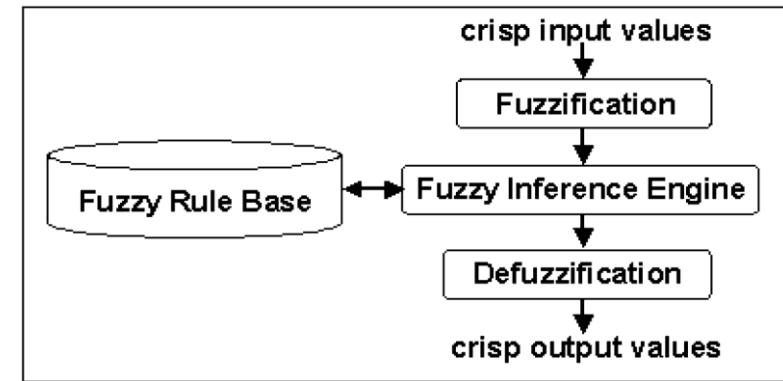
Zugehörigkeitsfunktionen

- Bestimmen die Zugehörigkeit eines Objekts zu einer Menge abhängig von seinen Attributen
- Beispiel: Zugehörigkeitsfunktionen für Short, Medium und Tall
 - Attribut: Höhe
 - Ein Objekt, dessen Höhe 1.55m beträgt, gehört zu 0.25 zu Short, zu 0.75 zu Medium und zu 0.0 zu Tall



Operatoren in Fuzzy Logik:

- OR: Wenn Fuzzy Statement A zu m wahr ist, und Fuzzy Statement B zu n , dann ist das Statement „A OR B“ zu k wahr, wobei $k = \max(m,n)$
- AND: Wenn Fuzzy Statement A zu m wahr ist, und Fuzzy Statement B zu n , dann ist das Statement „A AND B“ zu k wahr, wobei $k = \min(m,n)$
- NOT: Wenn Fuzzy Statement A zu m wahr ist, dann ist das Statement „NOT A“ zu k wahr, wobei $k = (1.0 - m)$



Regeln in Fuzzy Logik:

- Beispiel: “If Self is Tall and Enemy is Short, then Attack”
- Gegeben “Self” ist zu 30% Tall und “Enemy” ist zu 60% Short, dann ist die Bedingung zu 30% wahr. Angreifen oder nicht?

Defuzzification und Entscheidungsfindung

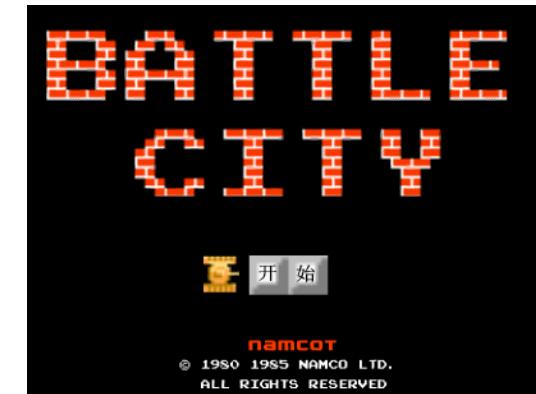
- Festlegen eines beliebigen Schwellwerts, z.B. 0.5. Wenn eine Bedingung zu mindestens diesem Schwellwert wahr ist, wird die Regel aktiviert
 - Im Beispiel sind es nur 0.3, wir würden also nicht angreifen
- Zufällige Entscheidung auf Basis des Wahrheitsgrads
 - Im Beispiel mit 30%iger Wahrscheinlichkeit angreifen
- Aber: Meistens gibt es mehrere Regeln, z.B.:
 - Rule 1: If Self is Tall and Enemy is Short, then Attack.
 - Rule 2: If Self is Big and Enemy is Green, then Attack.
 - Rule 3: If Self's Power is Great, and Self's Health is Okay, then Attack.
 - Mehrere Möglichkeiten:
 - Mehrheitsentscheid mit Schwellwert: Sind mindestens zwei Bedingungen zu mehr als 0.5 wahr, dann Angriff
 - Durchschnitt mit Schwellwert: Ist der Durchschnitt aller Bedingungen zu mehr als 0.5 wahr, dann Angriff
 - Zufällige Entscheidung auf Basis des normierten Wahrheitsgrads

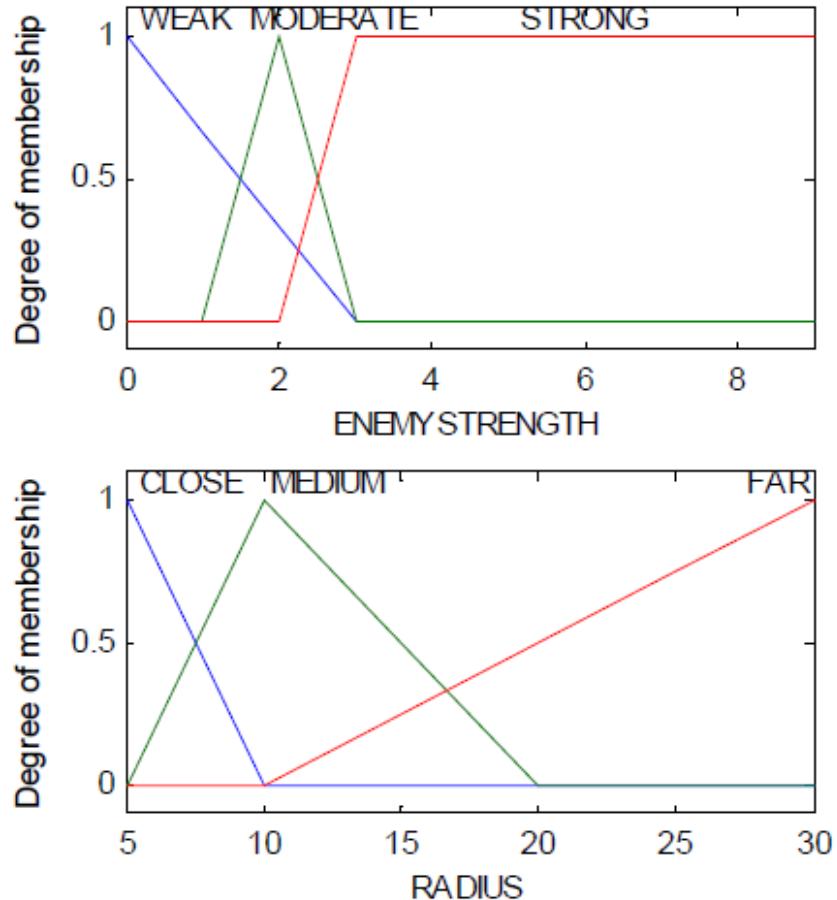
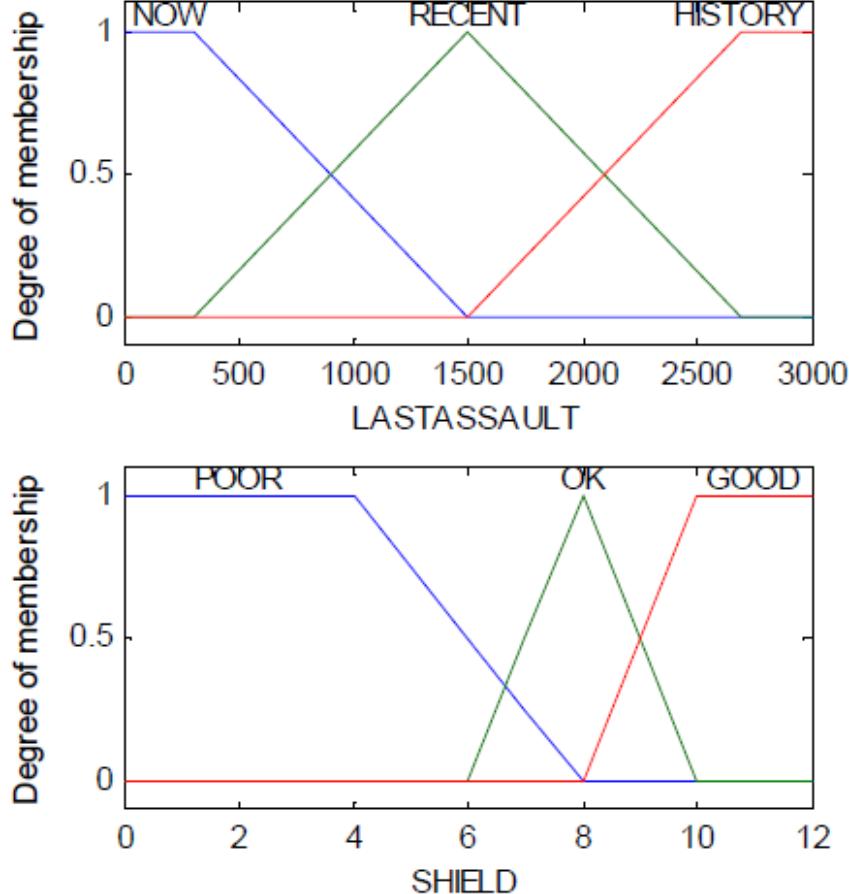
Li et al. „Fuzzy Logic in Agent-Based Game Design“ (2004)

Neuprogrammierung des NES-Spiels „BattleCity“ als „BattleCity.net“ mit KI-Spielern

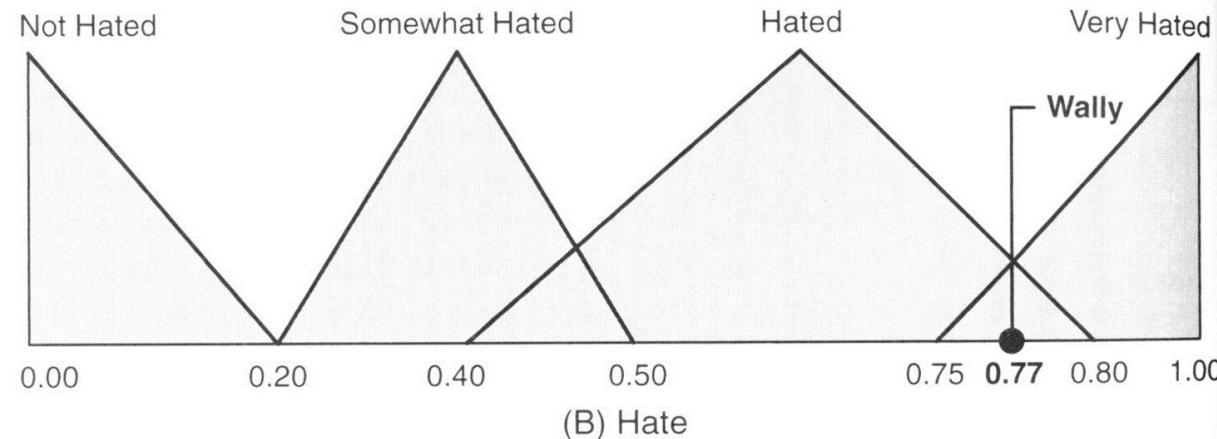
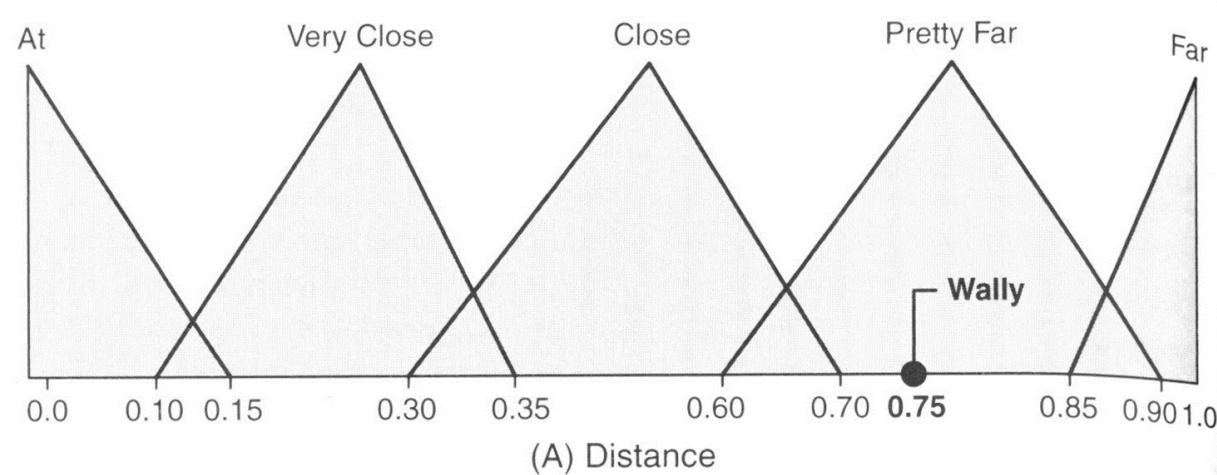
Ziel: Mit zwei Panzern gegnerische Basis zerstören, eigene Basis schützen

Ansatz verwendet Regelsystem und Fuzzy Logik um die eigenen Panzer zu dirigieren





Alexander: „An Optimized Fuzzy Logic Architecture for Decision Making“



Alexander: „An Optimized Fuzzy Logic Architecture for Decision Making“

	At	Very Close	Close	Pretty Far	Very Far
Not Hated	None	None	None	None	None
Somewhat Hated	Melee	Melee	Crossbow	None	None
Hated	Melee	Melee	Crossbow	Crossbow	Fireball
Very Hated	Melee	Melee	Crossbow	Fireball	Fireball

Combat Rules

IF Distance IS Pretty Far AND Hate IS Hated THEN use Crossbow behavior

IF Distance IS Pretty Far AND Hate IS Very Hated THEN use Fireball behavior



Bewertung von Fuzzy Logik

Vorteile:

- Erlaubt differenziertere Entscheidungen
- Gut bei vielen sich überschneidenden Regeln
- Lässt sich mit Wahrscheinlichkeiten kombinieren

Nachteile:

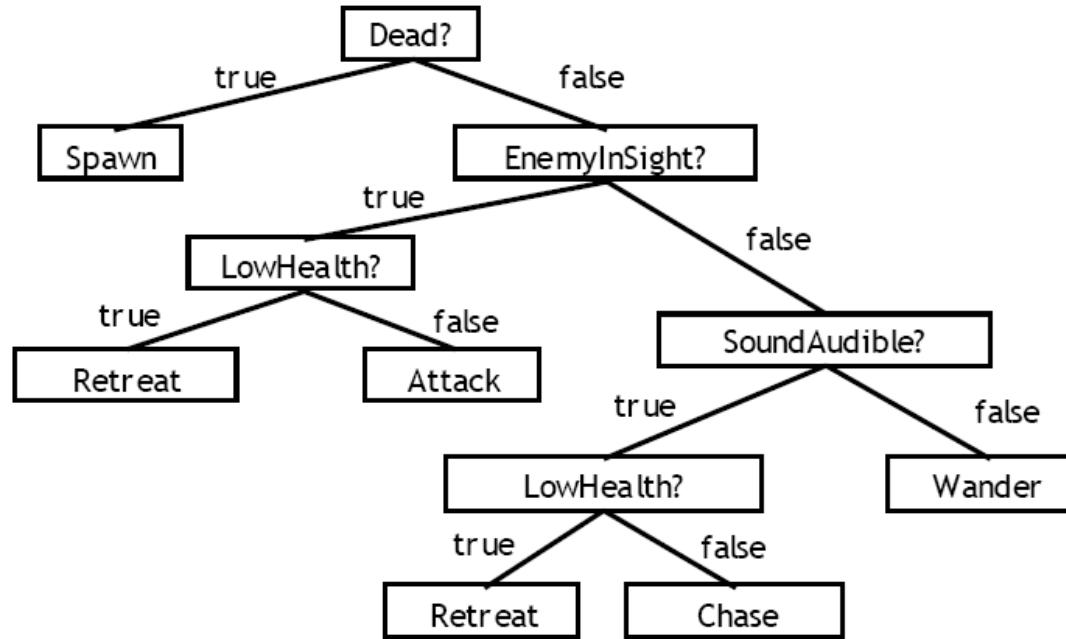
- Schwerer zu implementieren als diskrete Logik
- Erfordert höheren Rechenaufwand
- Design aufwändiger
- Mathematisches Wissen nötig

“I would agree that discrete logic is the default in game AI today.
However, I would also contend that it is also the primary weakness of game AI today.”

Dave Mark, Intrinsic Algorithm

- Überführung des Entscheidungsfindungsproblems in eine Klassifikationsaufgabe:
 - Zustand der Welt: Menge von Attributen
 - Für einen gegebenen Zustand gibt es eine geeignete Aktion (Klasse)
- Vorteil: Klassifikationsprobleme sind gut erforscht
- Entscheidungsbäume:
 - Knoten repräsentieren Attributtests
 - Blätter repräsentieren Klassen
 - Klassifizieren durch Absteigen von der Wurzel zu einem Blatt:
 - Führe an jedem Knoten den Test durch und steige den entsprechenden Ast hinab
 - Wird ein Blatt erreicht, gibt die Klasse, zu der das Blatt korrespondiert, zurück

Entscheidungsbaum für Quake

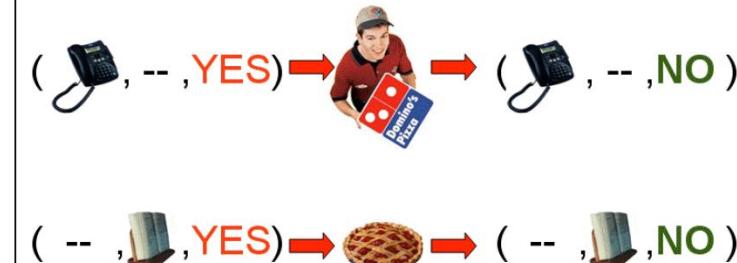


- Entscheidungsbäume sind kompakt und einfach zu verstehen
- Entscheidungsbäume können von Hand konstruiert werden (mühsam), werden aber in der Regeln gelernt (s. später)

- KI versucht, ein bestimmtes Ziel zu erreichen
- Welt kann verschiedene Zustände annehmen
- Zustände sind durch Eigenschaften definiert
- Ziel ist definiert als ein bestimmter Zustand der Welt
- KI kann Aktionen ausführen, die Zustände der Welt in andere überführen
- Aktionen haben Vor- und Nachbedingungen und können Kosten haben
- Planung ist das Finden einer (günstigsten) Folge von Aktionen, die den aktuellen Weltzustand in den gewünschten Zielzustand überführt
- Planung wird i.d.R. durch Anwendung von Graphenalgorithmen gelöst

State: (phone#, recipe, hungry?)

Goal: (-- , -- , NO)



STRIPS

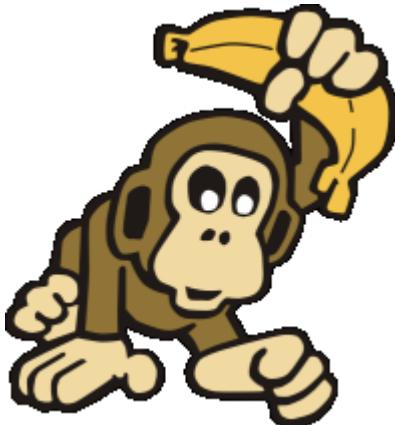
- **Stanford Research Institute Problem Solver**
- STRIPS bezeichnet:
 - Planer entwickelt von Fikes und Nilsson (1971)
 - Eine formale Sprache, in der der Input für diesen Planer ausgedrückt wird
- Dient auch heute noch als Grundlage für viele Planer und deren formale Sprachen
- Viele Implementierungen, meist in Prolog

STRIPS: Monkey & Banana

Initial state: $\text{At(A)}, \text{Level(low)}, \text{BoxAt(C)}, \text{BananasAt(B)}$

Goal state: Have(Bananas)

Actions:



Move(X, Y)

Preconditions: $\text{At}(X), \text{Level(low)}$

Postconditions: $\text{not At}(X), \text{At}(Y)$

ClimbUp(Location)

Preconditions: $\text{At}(\text{Location}), \text{BoxAt}(\text{Location}), \text{Level(low)}$

Postconditions: $\text{Level(high)}, \text{not Level(low)}$

ClimbDown(Location)

Preconditions: $\text{At}(\text{Location}), \text{BoxAt}(\text{Location}), \text{Level(high)}$

Postconditions: $\text{Level(low)}, \text{not Level(high)}$

MoveBox(X, Y)

Preconditions: $\text{At}(X), \text{BoxAt}(X), \text{Level(low)}$

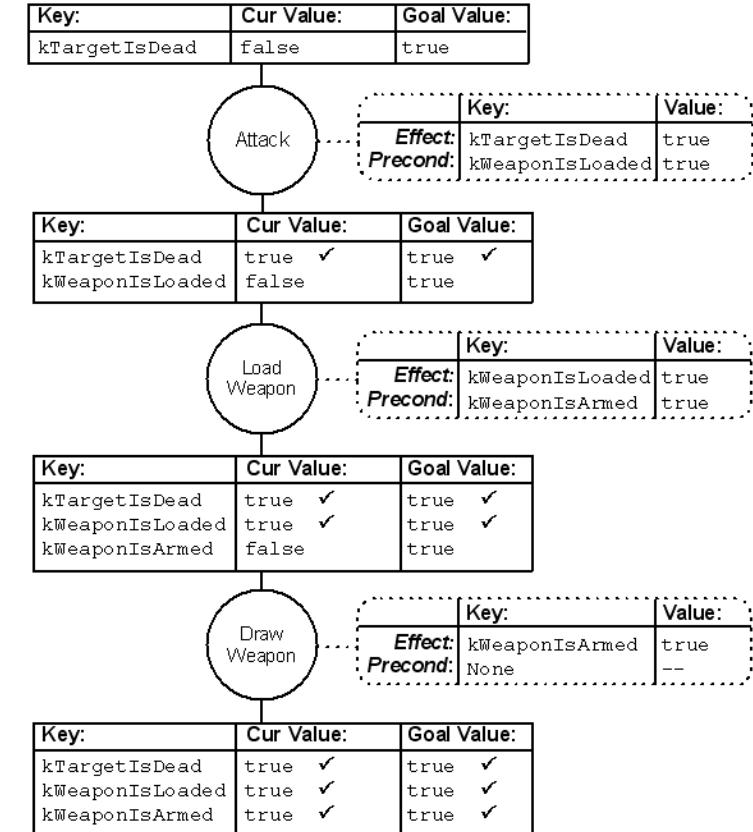
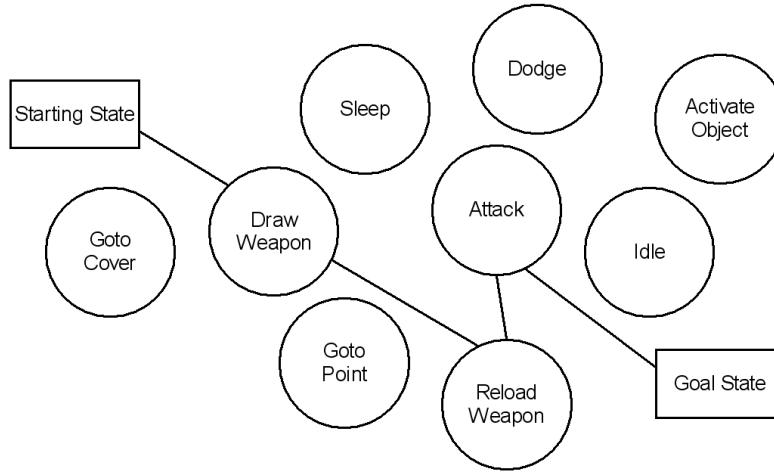
Postconditions: $\text{BoxAt}(Y), \text{not BoxAt}(X), \text{At}(Y), \text{not At}(X)$

TakeBananas(Location)

Preconditions: $\text{At}(\text{Location}), \text{BananasAt}(\text{Location}), \text{Level(high)}$

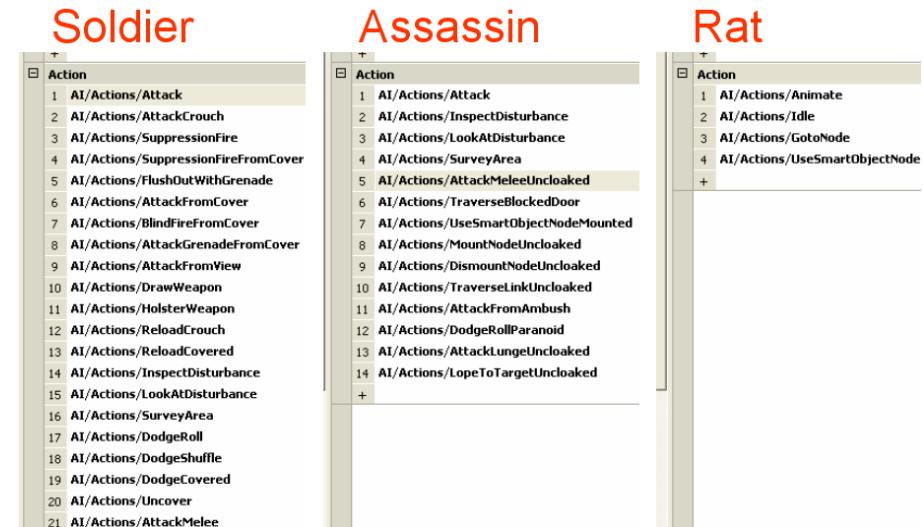
Postcondition: Have(bananas)

- Beispiel: Goal-Oriented Action Planning in F.E.A.R.
- Ziel: Gegner ausschalten



- Basierend auf STRIPS
- Plansuche regressiv (rückwärts) mit A*
- Weltzustandsspeicherung nicht als Tupel, sondern einzelne Eigenschaften

- Vorteile des Planens:
 - Sehr ausdrucksstark
 - Definition von intentionalen Verhaltensweisen auf deklarative Art und Weise
 - Dynamisches Problemlösen (Beispielvideos)
 - Erhöhte Wiederverwertbarkeit von Code, einfach zu erweitern
 - Einfache Möglichkeit, unterschiedliche Verhaltensweisen zu erzeugen, indem unterschiedliche Gegnerarten unterschiedliche Aktionen oder Kosten zugewiesen bekommen (Beispielvideos)
- Nachteile:
 - Integration in die „echte Welt“ nicht einfach





F.E.A.R. Ratte



F.E.A.R. Soldat



F.E.A.R. Assassine



F.E.A.R. Problemlösung I



F.E.A.R. Problemlösung II

- Ähneln Hierarchischen endlichen Automaten, aber: strenge Hierarchie keine Kreisläufe
- Nur Blätter beinhalten Aktionen, andere Knoten enthalten Regeln und dienen der Selektion von Kindknoten
- Behavior Trees werden zur Zeit als die Zukunft der KI in Spielen gehandelt, da sie alle Vorteile von Skripten, Hierarchischen EAs und Planen ohne deren Nachteile vereinen (d.h. den besten Kompromiss darstellen):
 - modular
 - Entscheidungen zur Laufzeit
 - Kontrolle und Überwachung der Aktionsausführung
 - übersichtlich und intuitiv

„In short they are the new, sexier finite state machines, basically a good way of laying out behaviors for your game agents in a nice modular and reusable way.“

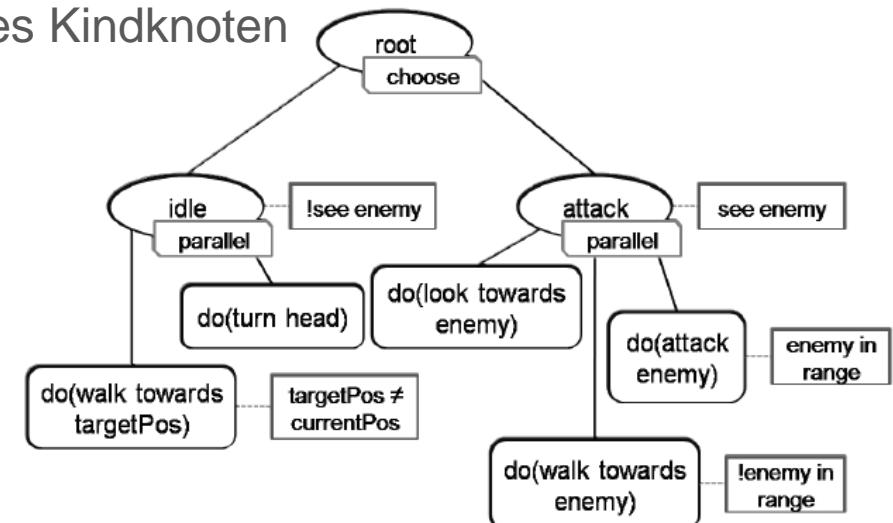
James Ford, Garage Games

Behavior Trees in Halo (Isla, 2005)

4 Möglichkeiten der Selektion von Kindknoten:

- Parallel Execution: Alle Kindknoten, deren Regeln zutreffen, werden gleichzeitig ausgeführt
- Choice: Derjenigen Kindknoten, der am besten zutrifft, wird ausgewählt
- Sequence: Alle Kindknoten, deren Regeln zutreffen, werden der Reihe nach ausgeführt
- Random: Zufällige Auswahl eines Kindknoten

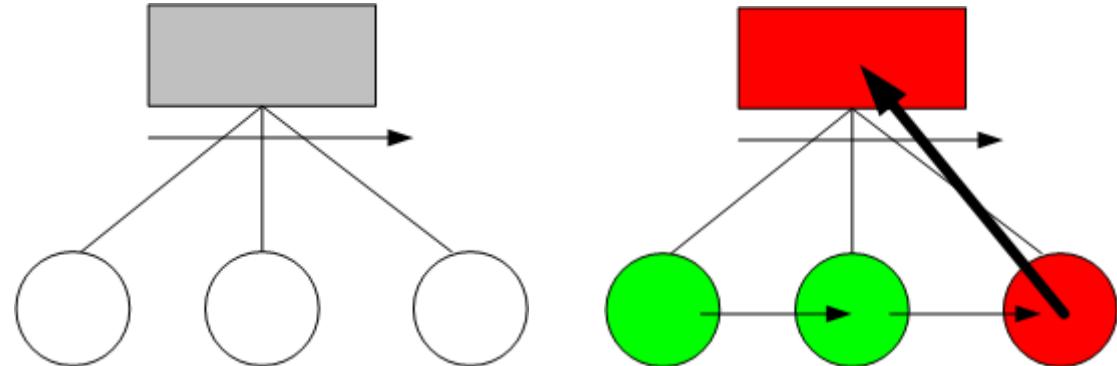
Einfaches Beispiel: Wache



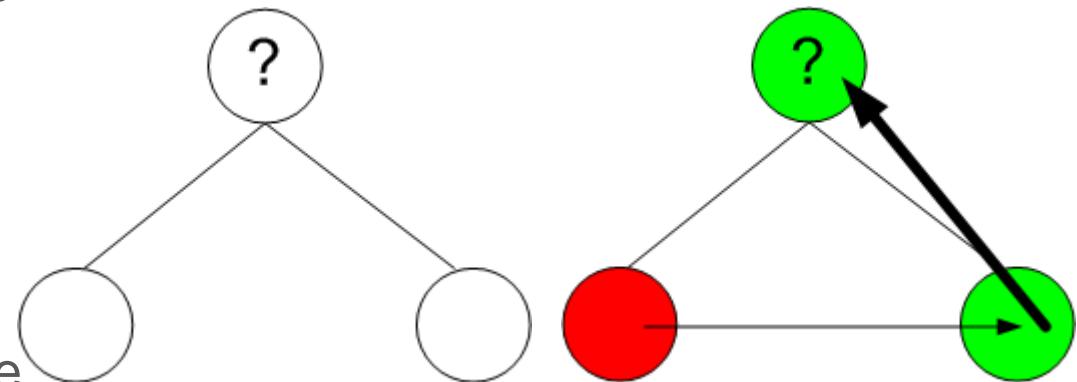
Neuer Ansatz (Champandard, <http://aigamedev.com/>, 2007):

- Blattknoten sind entweder Aktionen oder Überprüfungen
- Aktionen stellen Schnittstelle zur restlichen Engine dar, mögliche Aktionen:
 - Abspielen einer Animation
 - Spielen eines Sounds
 - Bewegung
 - Interaktion mit der Umgebung
 - Abfragen / Verändern von Werten
 - ...
- Nicht-Blatt-Knoten sind Entscheidungsregeln oder dienen zur Ablaufsteuerung: Sequence, Selector, Parallel, Decorator
- Jeder Knoten liefert Auskunft über Erfolg oder Misserfolg der Aktion an Elternknoten zurück

- Sequence („AND“)
 - Aktionen nacheinander
 - strenge Reihenfolge
 - Abbruch bei False
 - Fortsetzung bei True
 - True, falls alle UB True
 - Ausführung zusammengesetzter Aktionen

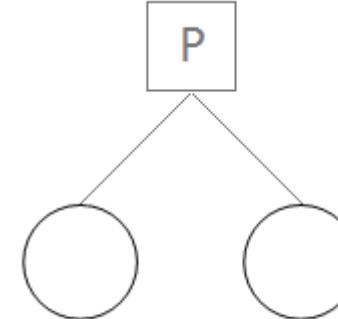


- Selector („OR“)
 - Auswahl einer Aktion
 - Abbruch bei True
 - Fortsetzung bei False
 - False, falls alle UB False
 - Lokales Finden von Alternativen



- **Parallel**

- Parallele Ausführung der UBs
- True, wenn alle UB True
- evtl. Parallelitätsprobleme

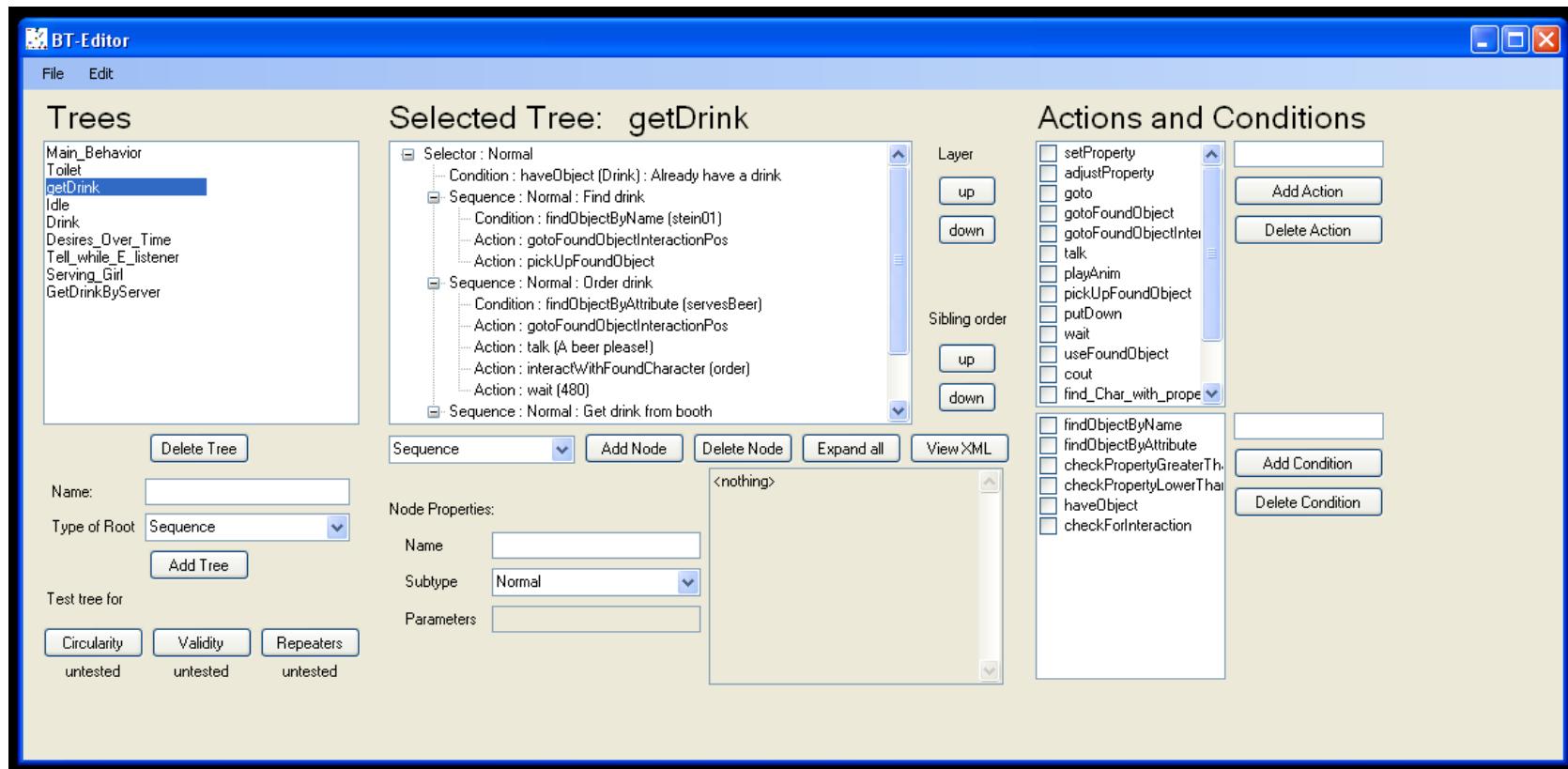


- **Decorator**

- Sind anderen Knoten vorgeschaltet und beeinflussen deren Ausführung
- Lookup (Modularität)
- Filter
- Schleifen
- Limits
- Semaphore
- ...



Behavior Trees in der Horde3D GameEngine (Diplomarbeit F. Bujtor)



Var and the Vikings: Educational Game um Programmieren zu Lernen



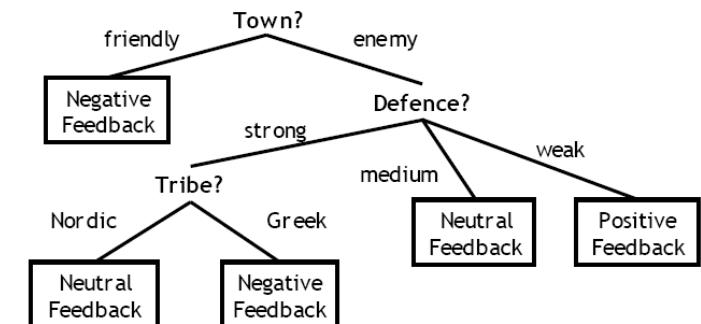
http://var.i.brainworth.net/main.html?level=introduction_to_actions_moveforward



Lernverfahren

- Werden anhand von Beispielen gelernt
- Beispiele: Zustand der Welt, dazugehörige Entscheidung
- Algorithmus konstruiert Baum anhand von Beispielen
- Beispiel: Black & White, welche Dörfer darf die Kreatur angreifen?

Nr.	Town	Defence	Tribe	Feedback from Player
1	Friendly	Weak	Celtic	Negative
2	Enemy	Weak	Celtic	Positive
3	Friendly	Strong	Norse	Negative
4	Enemy	Strong	Norse	Neutral
5	Friendly	Medium	Greek	Negative
6	Enemy	Medium	Greek	Neutral
7	Enemy	Strong	Greek	Negative
8	Enemy	Medium	Aztec	Neutral
9	Friendly	Weak	Aztec	Negative



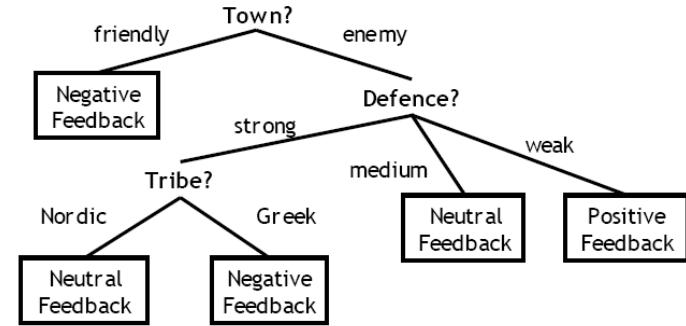
Entscheidungsbäume

- ID3 Algorithmus

```

Function induce_tree(trainingSet,props){
    IF all entries in trainingSet are in the same class
    THEN return a leaf node labeled with that class
    ELSE IF props is empty
        THEN RETURN leaf node labeled with disjunction of
            all classes in trainingSet
    ELSE select a property P as root of current tree;
        delete p from props;
        FOR EACH value V OF P
            Create a branch of tree labeled with V;
            Cv := {x ∈ trainingSet: P(x)=V};
            induce_tree(Cv,props),
            attach results to branch V}
    
```

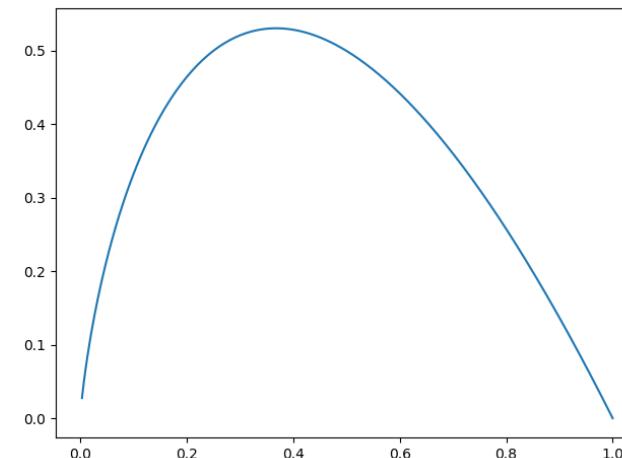
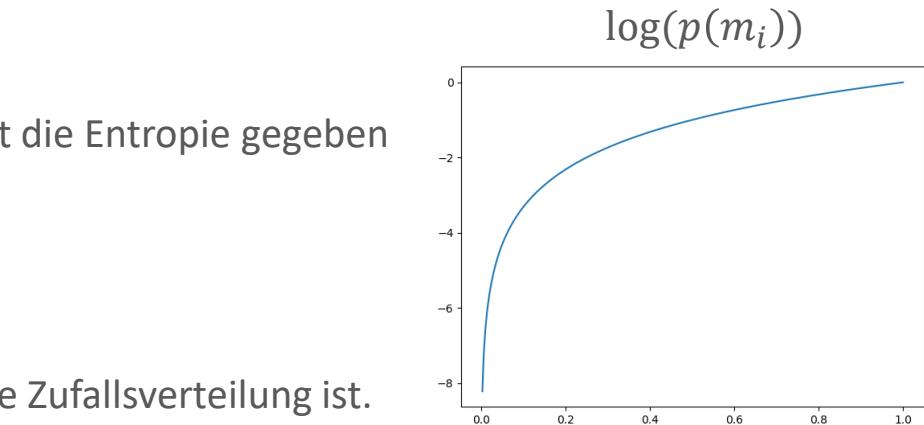
Nr.	Town	Defence	Tribe	Feedback from Player
1	Friendly	Weak	Celtic	Negative
2	Enemy	Weak	Celtic	Positive
3	Friendly	Strong	Norse	Negative
4	Enemy	Strong	Norse	Neutral
5	Friendly	Medium	Greek	Negative
6	Enemy	Medium	Greek	Neutral
7	Enemy	Strong	Greek	Negative
8	Enemy	Medium	Aztec	Neutral
9	Friendly	Weak	Aztec	Negative



- Auswahl der nächsten Eigenschaft: Eigenschaft, die den höchsten Informationsgewinn bietet
- Um den Informationsgehalt zu messen verwenden wir Entropie.

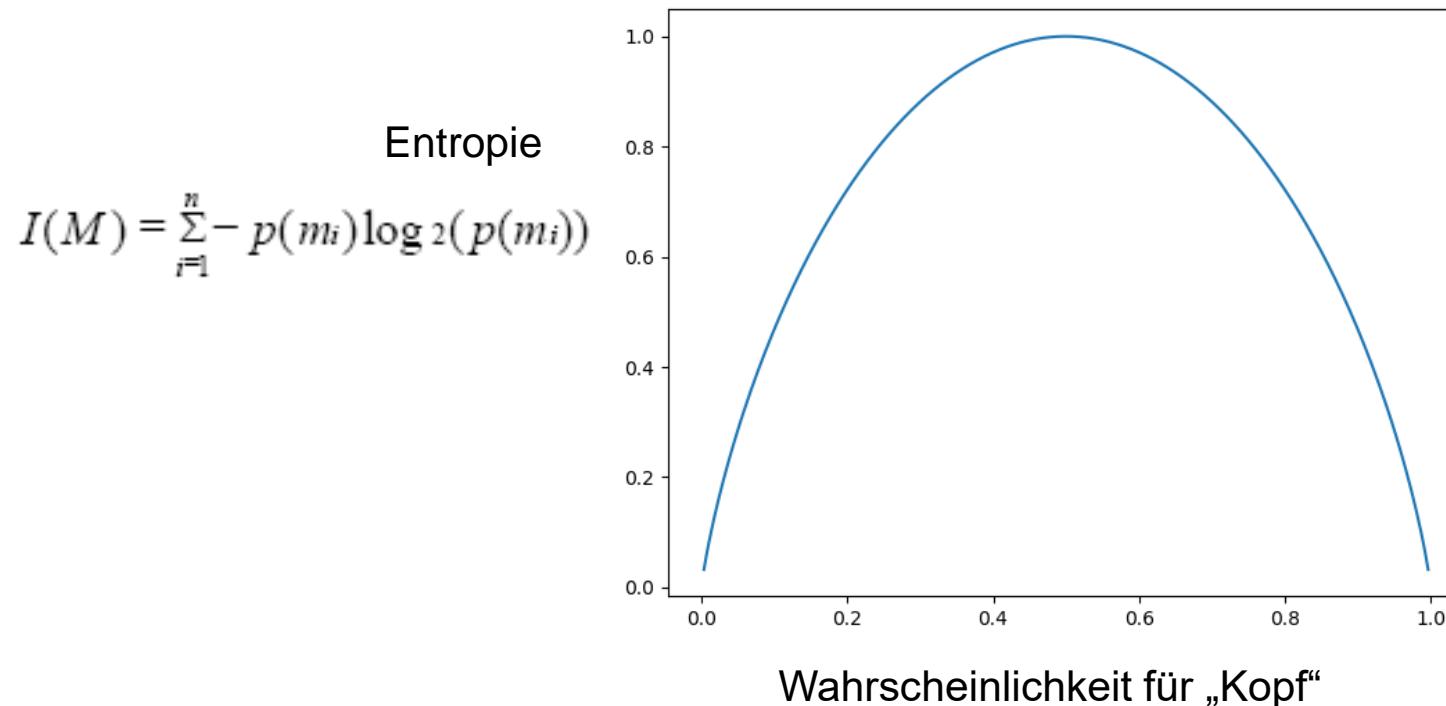
- ID3 Algorithmus
- Für eine Zufallsverteilung $M=\{m_1, \dots, m_n\}$ ist die Entropie gegeben durch: $I(M) = \sum_{i=1}^n -p(m_i)\log_2(p(m_i))$
- Die Entropie ist höher, je unvorhersehbarer die Zufallsverteilung ist.
- Das sieht man schon daran, dass jeder Summand $-p(m_i)\log(p(m_i))$ genau dann hoch ist, wenn $p(m_i)$ sehr „zufällig“ eintritt

$$-p(m_i)\log(p(m_i))$$



$$p(m_i)$$

- ID3 Algorithmus
- Beispiel:
- Für einen gezinkten Münzwurf, bei dem die Wahrscheinlichkeit der beiden Ergebnisse nicht unbedingt gleich ist, verhält sich die Entropie wie folgt:



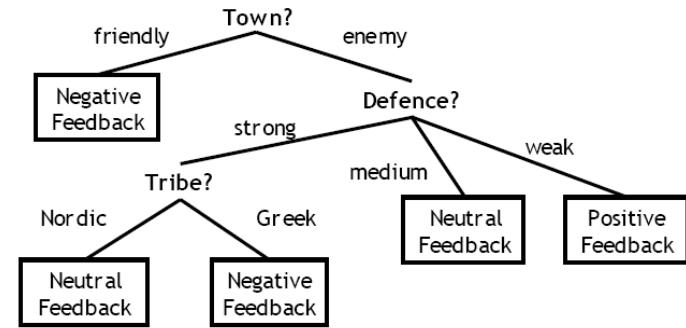
Entscheidungsbäume

- ID3 Algorithmus

```
Function induce_tree(trainingSet,props){
    IF all entries in trainingSet are in the same class
    THEN return a leaf node labeled with that class
    ELSE IF props is empty
        THEN RETURN leaf node labeled with disjunction of
            all classes in trainingSet
    ELSE select a property P as root of current tree;
        delete p from props;
        FOR EACH value V OF P
            Create a branch of tree labeled with V;
            Cv := {x ∈ trainingSet: P(x)=V};
            induce_tree(Cv,props),
            attach results to branch V}
}
```

- Formel für Entropie: $I(M) = \sum_{i=1}^n -p(m_i)\log_2(p(m_i))$
- Sei C die Menge aller Trainingsdaten (in unserem Fall also die gesamte Tabelle).
- Wählt man Eigenschaft P mit n Werten v₁ ... v_n als Wurzel des aktuellen Baums, dann wird C in die Teilmengen {C₁,...,C_n} aufgeteilt, wobei C_i = {x:P(x)=v_i} (P auch Attribut mit Ausprägung v₁)
- Entropie eines Baums mit Eigenschaft P als neue Wurzel: $E(P) = \sum_{i=1}^n \frac{|C_i|}{|C|} I(C_i)$
- Durch Eigenschaft P gewonnener Informationsgewinn: $gain(P) = I(C) - E(P)$

Nr.	Town	Defence	Tribe	Feedback from Player
1	Friendly	Weak	Celtic	Negative
2	Enemy	Weak	Celtic	Positive
3	Friendly	Strong	Norse	Negative
4	Enemy	Strong	Norse	Neutral
5	Friendly	Medium	Greek	Negative
6	Enemy	Medium	Greek	Neutral
7	Enemy	Strong	Greek	Negative
8	Enemy	Medium	Aztec	Neutral
9	Friendly	Weak	Aztec	Negative



Beispiel Informationsgewinn

- Informationsgehalt der Beispieltabelle:

$$I(C) = -\frac{5}{9} \log_2 \frac{5}{9} - \frac{3}{9} \log_2 \frac{3}{9} - \frac{1}{9} \log_2 \frac{1}{9} \approx 1.352$$

- Teste Town als Wurzel des Baums:
 $C_1 = \{1,3,5,9\}$, $C_2 = \{2,4,6,7,8\}$
- Informationsgewinn durch Town:

$$E(Town) = \frac{4}{9} I(C_1) + \frac{5}{9} I(C_2) = \frac{4}{9} * 0.0 + \frac{5}{9} * 1.371 = 0.762$$

$$gain(Town) = I(C) - E(Town) = 1.352 - 0.762 = 0.59$$

- Nachteile des Lernens von Entscheidungsbäumen:
 - Notwendigkeit von Trainingsbeispielen
 - Gelernte Bäume können falsch sein

Nr.	Town	Defence	Tribe	Feedback from Player
1	Friendly	Weak	Celtic	Negative
2	Enemy	Weak	Celtic	Positive
3	Friendly	Strong	Norse	Negative
4	Enemy	Strong	Norse	Neutral
5	Friendly	Medium	Greek	Negative
6	Enemy	Medium	Greek	Neutral
7	Enemy	Strong	Greek	Negative
8	Enemy	Medium	Aztec	Neutral
9	Friendly	Weak	Aztec	Negative



- Neuronale Netze

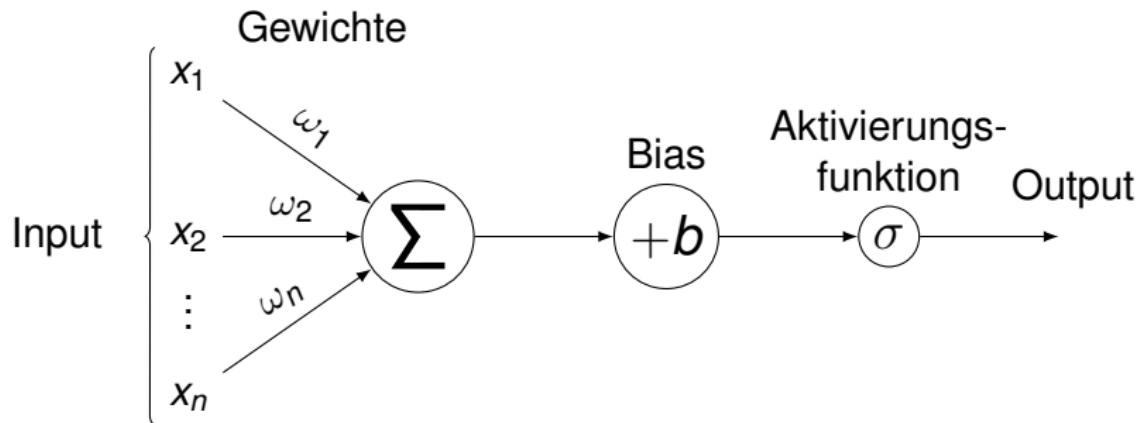


Neuronale Netze sind gerichtete Graphen (Netze), aus Neuronen.

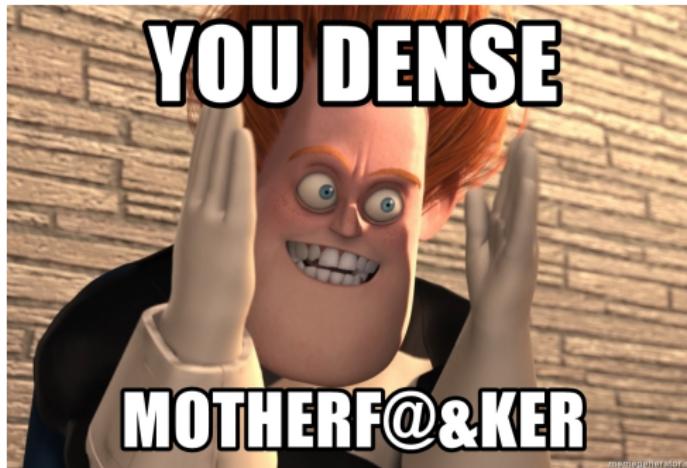
Definition

Ein **Neuron** ist eine Funktion $f : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$ der Form

$$f(x; \omega, b) = \sigma(\omega \cdot x + b)$$



Die einfachsten Neuronalen Netze bestehen nur aus "vollständig verbunden" und nach vorne gerichteten Neuronen. Sie heißen Dichte Netze (manchmal auch Feedforward Netze) und bestehen aus Perzeptronen/Dichten Schichten.

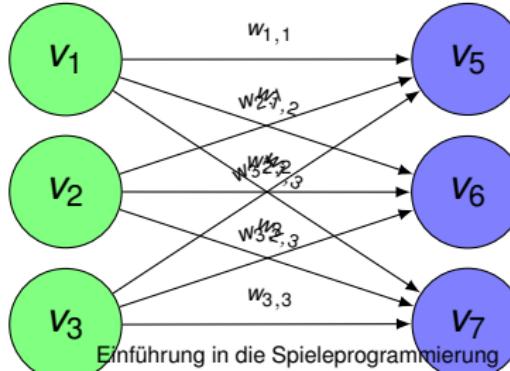


Definition

Ein **Perzepron** ist eine Funktion $P : \mathbb{R}^n \times (\mathbb{R}^{n \times m} \times \mathbb{R}^m) \rightarrow \mathbb{R}^m$ mit

$$P(x; W, b) = \sigma(Wx + b),$$

wobei σ die komponentenweise Anwendung einer Aktivierungsfunktion $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ beschreibt und $W = [w_{ij}]$ und b die Gewichtsmatrix und der Bias-Vektor sind.



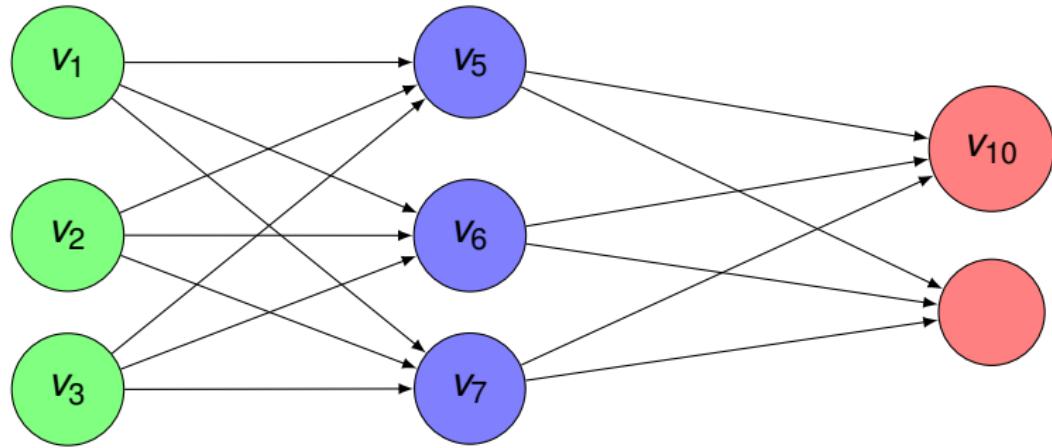


Definition

Ein **Multi Layer Perzepron(MLP)** mit n (Hidden) Layern ist eine Komposition aus $n + 1$ Perzeptronen $P_n(x, \theta_n)$. Das heißt ein MLP ist eine Funktion

$$f(x; \theta) = P_n(P_{n-1}(P_{n-2}(\dots); \theta_{n-1}); \theta_n).$$

Hierbei bezeichnet θ_i jeweils das Tupel (W_i, b_i) aus der Gewichtsmatrix und dem Bias-Vektor des jeweiligen Perzeptrons.



Input x

$$P_1(x, \theta_1) \\ := f_1(x)$$

$$P_2(P_1(x, \theta_1), \theta_2) \\ = P_2(f_1(x), \theta_2) \\ := f_2(x)$$

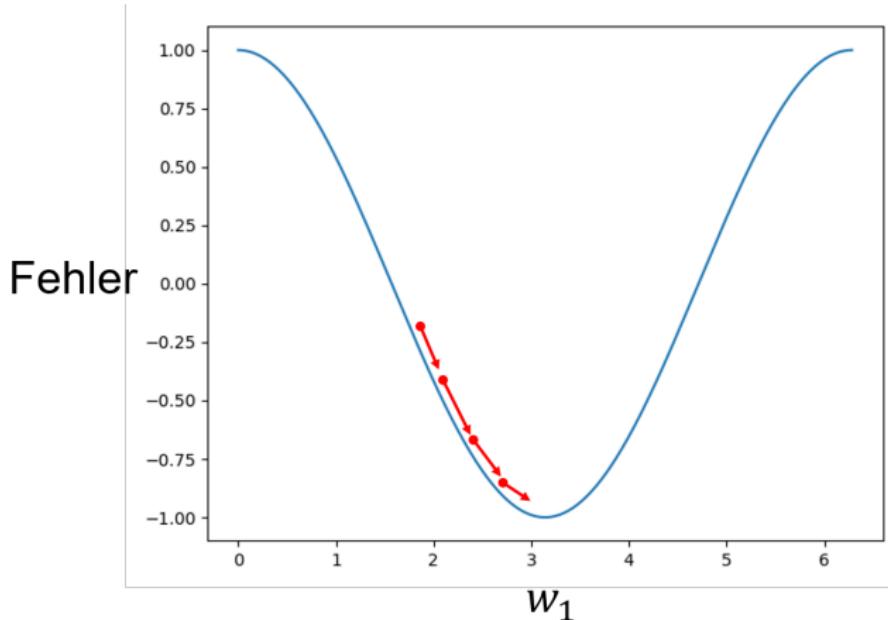


- NN ist von Parametern θ abhängige Funktion $f(\theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m$
- Grundlegende Idee: Minimierung einer Fehlerfunktion, die den Abstand zur *Grundwahrheit* $y \in \mathbb{R}^m$ misst, zum Beispiel *mean squared error*

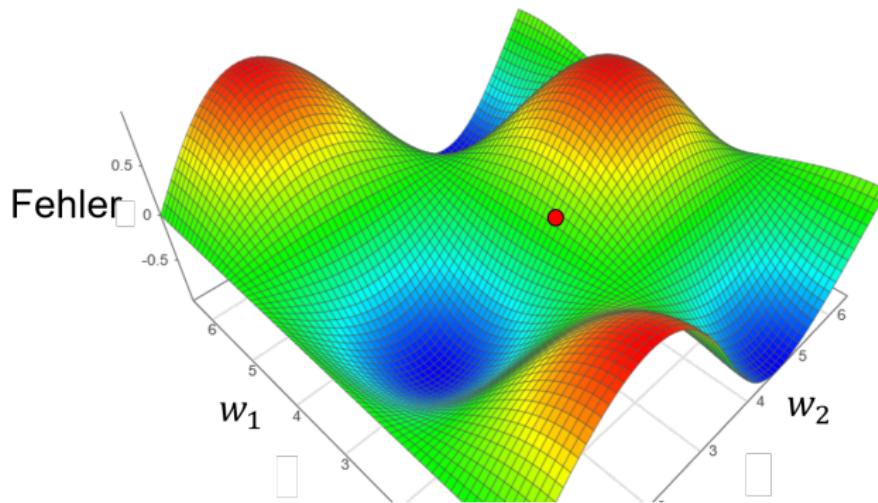
$$L(\theta) = \frac{1}{m} \sum_{i=1}^m (y_i - f(\theta)_i)^2$$

Dazu verwendet man (Stochastic) Gradient Descent

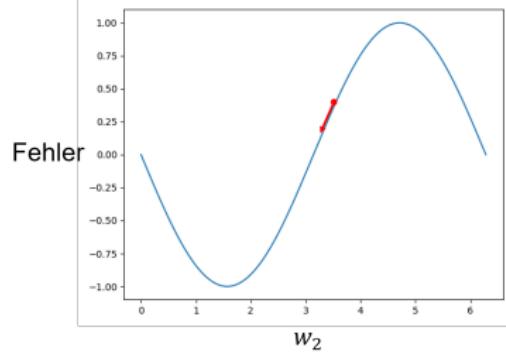
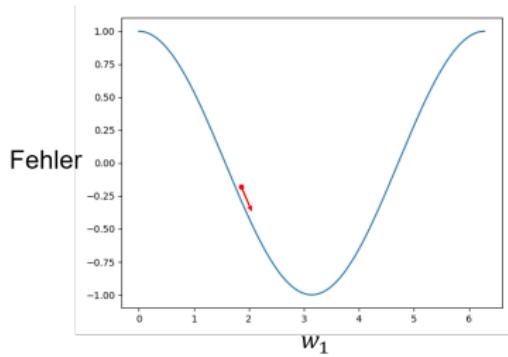
$$\theta_{i+1} = \theta_i - \alpha \nabla L(\theta_i)$$



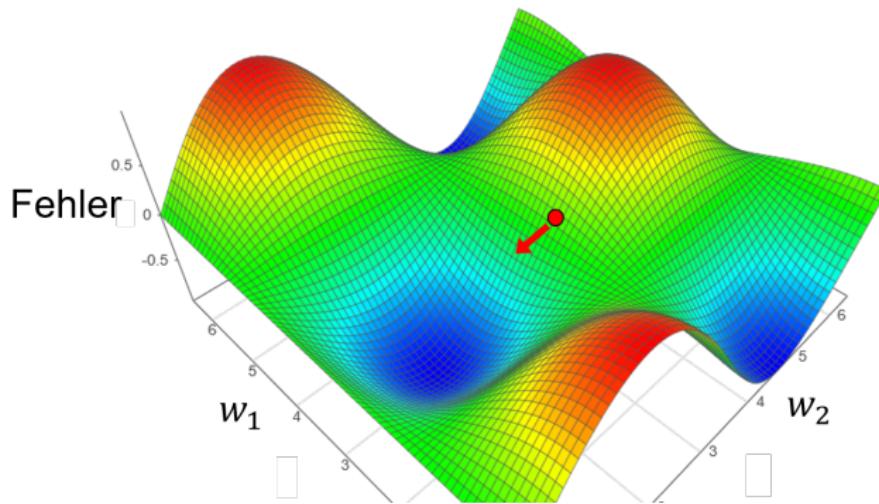
Mehrdimensionale Fehler Funktion:



Auch für jedes Gewicht wie im eindimensionalen Fall:



Und dann insgesamt:





Warum Backpropagation?

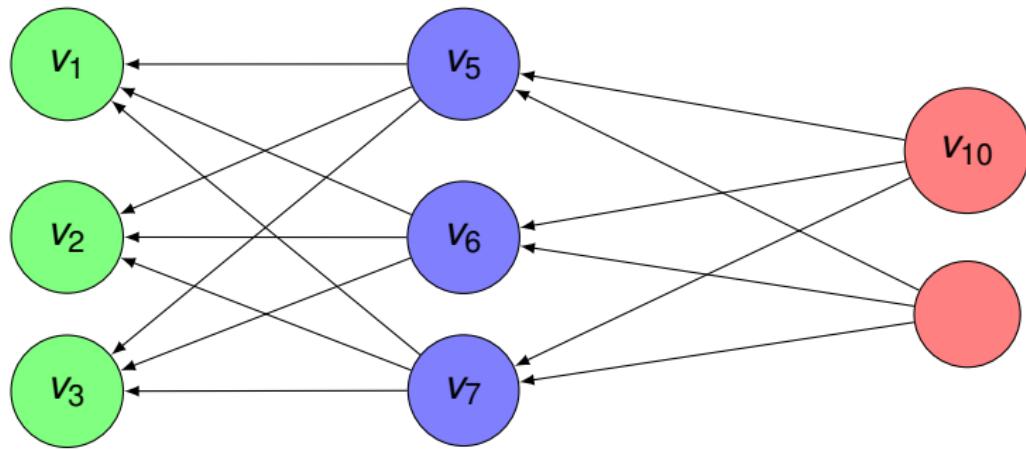
→ Der Fehler wird durch die Kettenregel durch das Netz zurückgegeben/zurückpropagiert.

Definition

Seien $f, g : \mathbb{R} \rightarrow \mathbb{R}$ zwei Funktionen, dann gilt für die Ableitung ihrer Komposition an der Stelle $x_0 \in \mathbb{R}$:

$$\frac{\partial f(g(x))}{\partial x}(x_0) = Df(g(x_0)) \cdot \frac{\partial g(x)}{\partial x}(x_0)$$

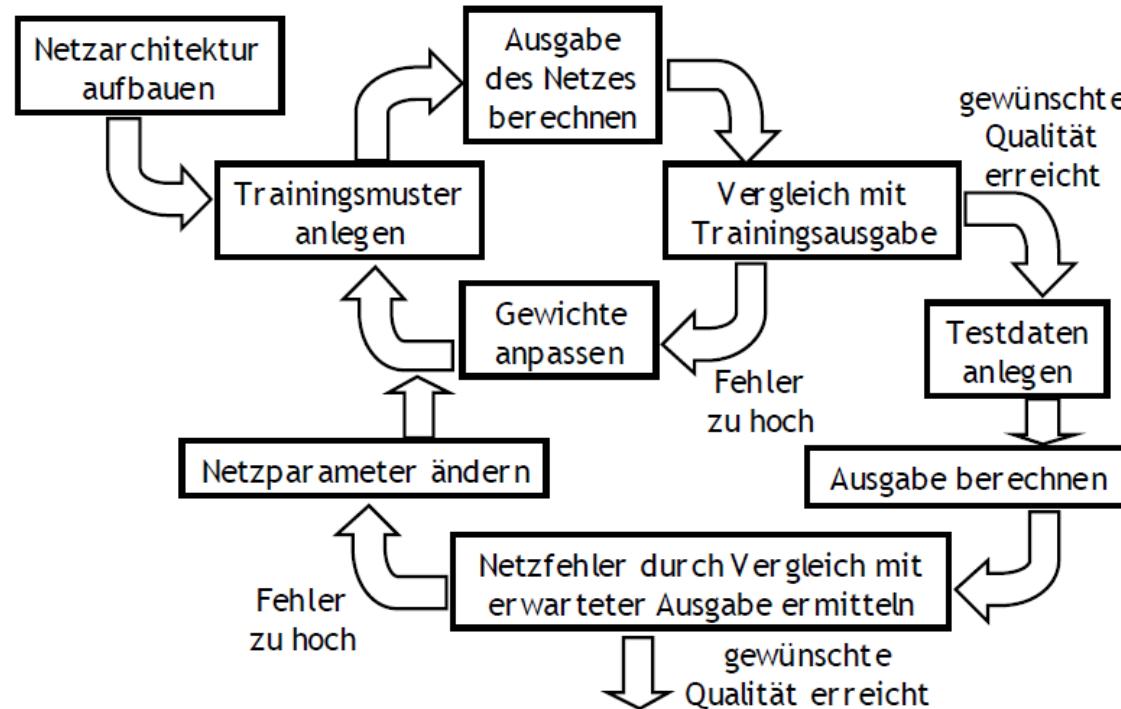
Hierbei bezeichnet $Df(g(x_0))$ die Jakobi-Matrix an der Stelle x_0 .



Fehler:

$$\frac{\partial P_1}{\partial w_{ij}}(x_0) Df_2(x_0) \nabla L(f_2(x_0)) \quad \frac{\partial P_2}{\partial w_{ij}}(x_0) \nabla L(f_2(x_0))$$

Netzentwicklung



Bewertung von Neuronalen Netzen

Vorteile:

- Hohe Geschwindigkeit durch massive Parallelität
- Geringer Entwicklungsaufwand
- Flexibilität
- Hohe Fehlertoleranz gegen Ausfall einzelner Neuronen
- Robust gegen Rauschen und unvollständige Eingabe

Nachteile:

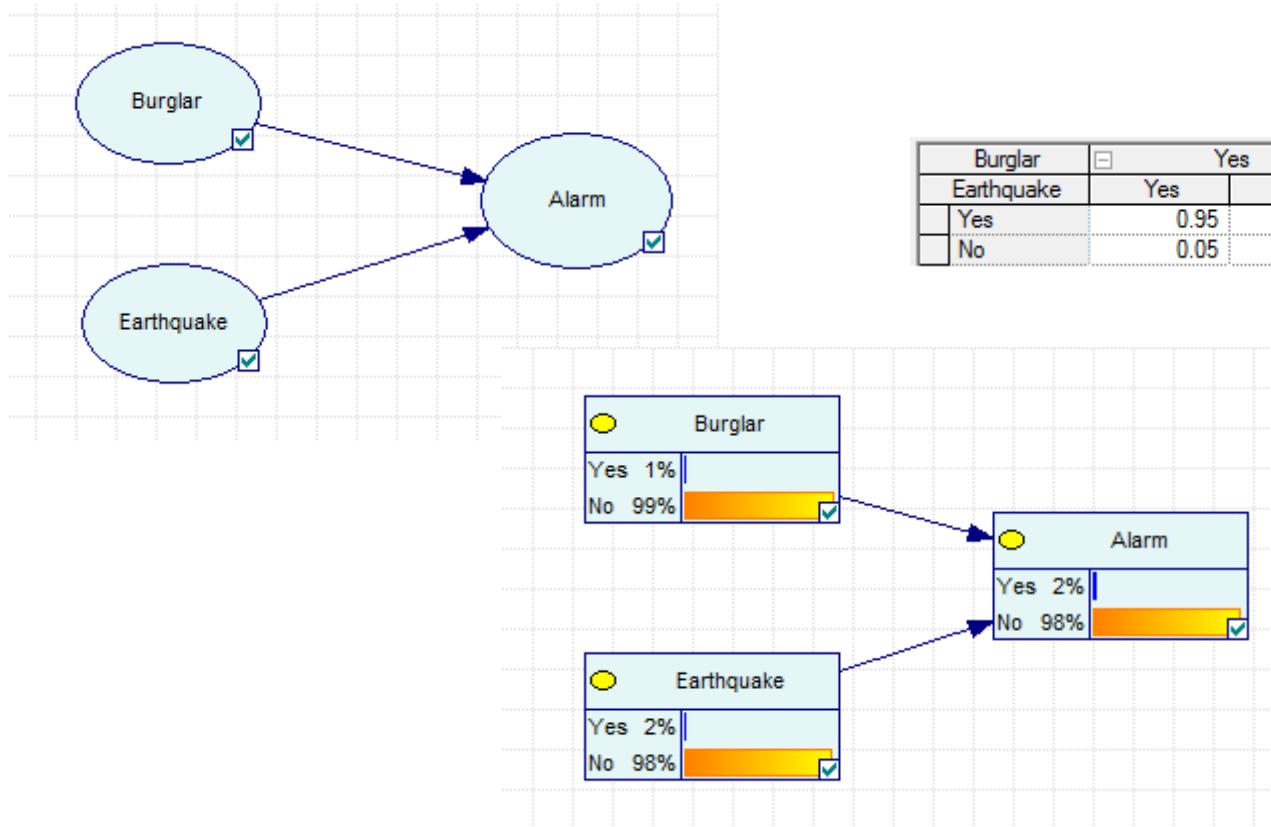
- Schwer zu verstehen
- Nicht immer intuitiv
- Lange Trainingszeiten, da viel Trainingsdaten erforderlich
- Validierung meist nicht möglich, ggf. suboptimale Lösungen

Hintergrund:

- Bedingte Wahrscheinlichkeit:
 $P(A|B) = P(A,B) / P(B)$
- Bayes' Theorem zur Umkehrung bed. Wahrscheinlichkeit:
 $P(A|B) = P(B|A) * P(A) / P(B)$

Bayes'sches Netz:

- Knoten stellen Zufallsvariablen (z.B. Ereignisse) dar
- Kanten stellen bedingte Abhängigkeiten dar
- Schlussfolgerungen über Propagation innerhalb des Netzes
- „Schlussfolgern im Ungewissen“
- Abhängigkeiten und Wahrscheinlichkeiten entweder vom Experten modelliert oder aus statistischen Daten gewonnen (Expectation-Maximization-Algorithmus)

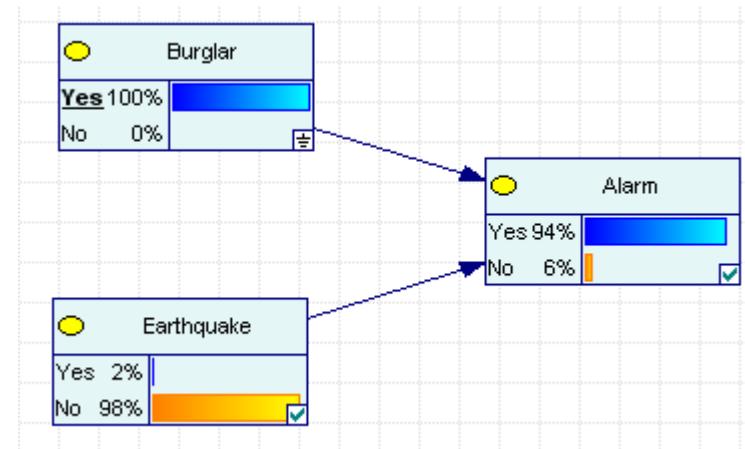


Alle Beispiele aus: Tozour – Introduction to Bayesian Networks and Reasoning Under Uncertainty

3 Arten der Schlussfolgerung

Kausal: Von der Ursache zum Effekt

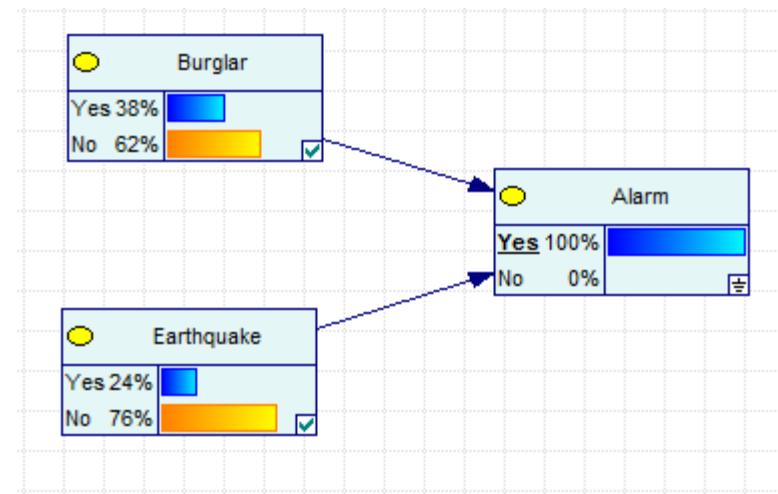
Wenn A B bedingt und A bekannt ist, kann B berechnet werden



3 Arten der Schlussfolgerung

Diagnostisch: Vom Effekt zur Ursache

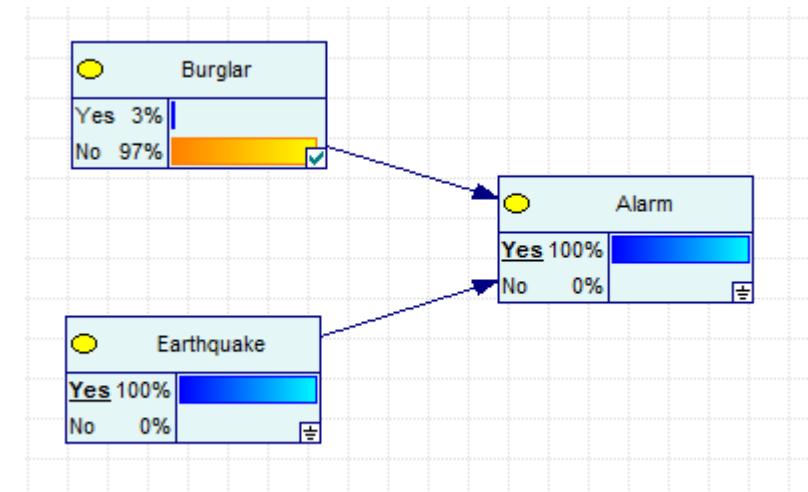
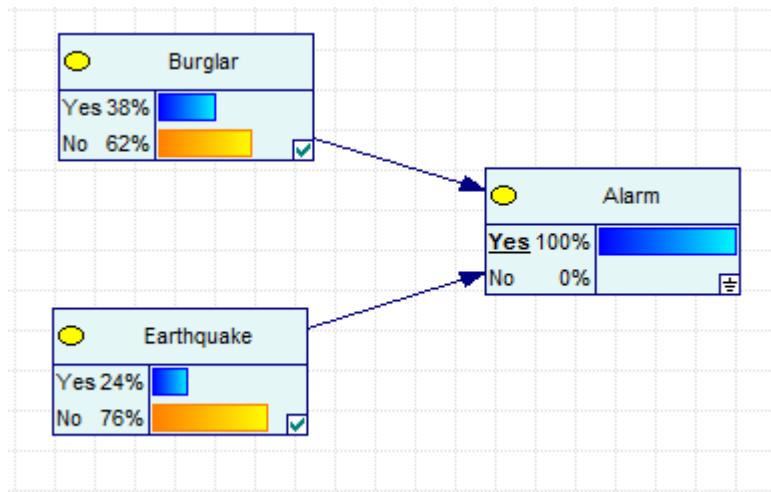
Wenn A B bedingt und B bekannt ist, kann A berechnet werden



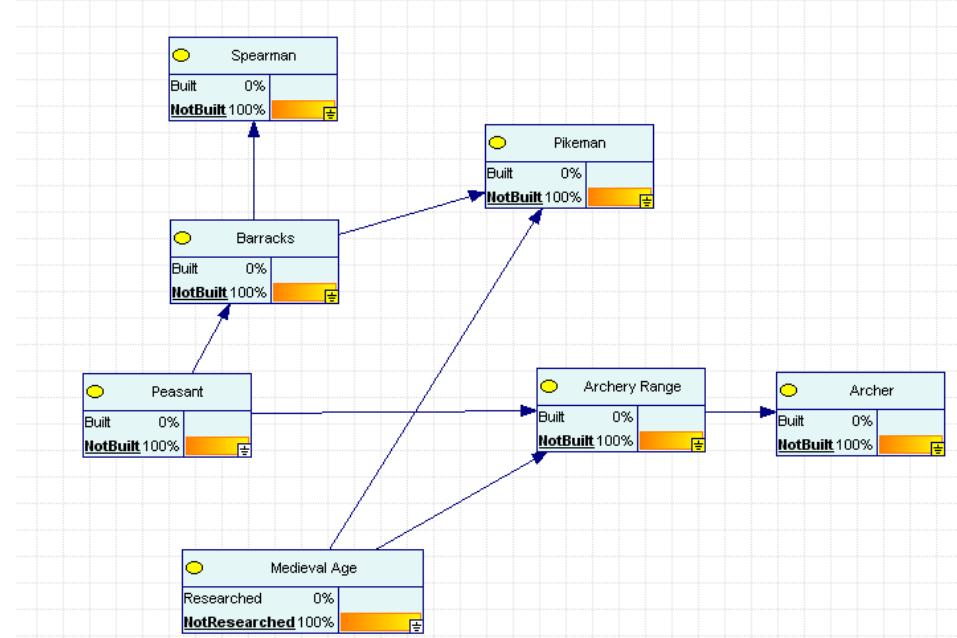
3 Arten der Schlussfolgerung

Interkausal: Zwischen Ursachen eines gemeinsamen Effekts

Wenn A und B C bedingen und C bekannt ist, kann berechnet werden, wie A B verändert, obwohl sie ursprünglich unabhängig waren



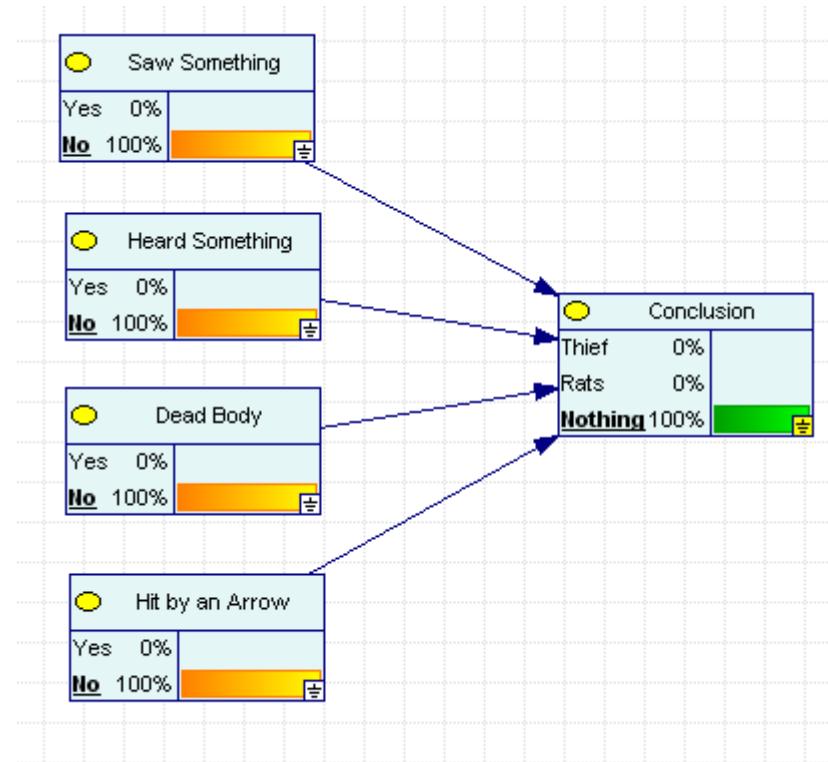
Anwendung in Spielen: Technologiebaum



Kausal: Wenn wir wissen, welche Gebäude / Technologien der Gegner gebaut / erforscht hat, können wir ableiten, welche Einheiten er hat.

Diagnostisch: Wenn wir wissen, welche Einheiten der Gegner gebaut hat, können wir ableiten, welche Gebäude / Technologien er gebaut / erforscht hat

Anwendung in Spielen: Wahrnehmung einer Wache



Genie & Smile: <http://genie.sis.pitt.edu/>

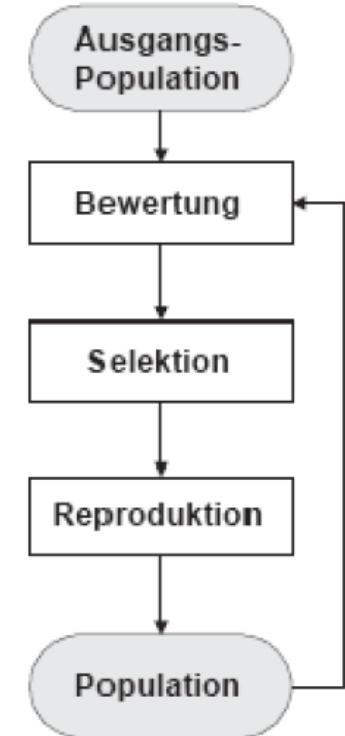


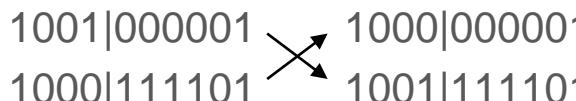
Genetische Algorithmen

- Inspiriert durch Darwin's Evolutionstheorie:
 - Entstehung zunehmend fähigerer Einzelwesen nach dem Prinzip der natürlichen Auslese
- Übertragung evolutionärer Konzepte, um Probleme zu lösen:
 - Erzeugung zunehmend fähigerer Problemlösungen durch die Verarbeitung von Populationen mit Problemlösungskandidaten
- Codierung des Problems als String (DNA)
- Suchraum: Wertekombinationen aller Attribute
- Eignung eines Chromosoms zur Lösung beizutragen wird durch Fitnessfunktion berechnet
- Geeignet für: komplexe Such- oder Optimierungsprobleme
- Anwendung in Spielen: Finden und Optimieren von Parametereinstellungen, Verhaltensmodellen und Wegen

Vorgehen:

- 1. Start: Kodiere Problem und generiere Population
- 2. Fitness: Berechne die Fitness $f(x)$ jedes Chromosoms
- 3. Nächste Population:
 - Selektion: Wähle potentielle Eltern ihrer Fitness nach aus
 - Rekombination: Kreuze nach einer bestimmten Wahrscheinlichkeit diese Eltern, um Nachwuchs zu erzeugen. Falls keine Kreuzung erfolgt, soll der Nachwuchs eine exakte Kopie der Eltern sein
 - Mutation: Mutiere den Nachwuchs nach einer bestimmten Wahrscheinlichkeit an einer Stelle
- 4. Wiederhole ab 2. bis die Abbruchbedingung erreicht ist und gebe bestes Chromosom aus der Population zurück



- Bewertung von Chromosomen:
 - Die Bewertungsfunktion eines genetischen Algorithmus legt das Optimierungskriterium und das Ziel des Algorithmus fest
 - Sie misst die Nähe eines Chromosoms zum Optimum
- Kreuzung:
 - Durch Kreuzung werden neue Lösungen erzeugt, die Komponenten ihrer Eltern in sich vereinigen
 - Unterschiedliche Kreuzungsoperatoren:
 - 1-point crossover:


The diagram shows two binary strings: "1001|000001" and "1000|000001". A vertical line separates the first four bits from the last five. An arrow points from the fifth bit of the first string to the fifth bit of the second string, indicating a crossover point at the fifth bit.

1001|000001 1000|000001

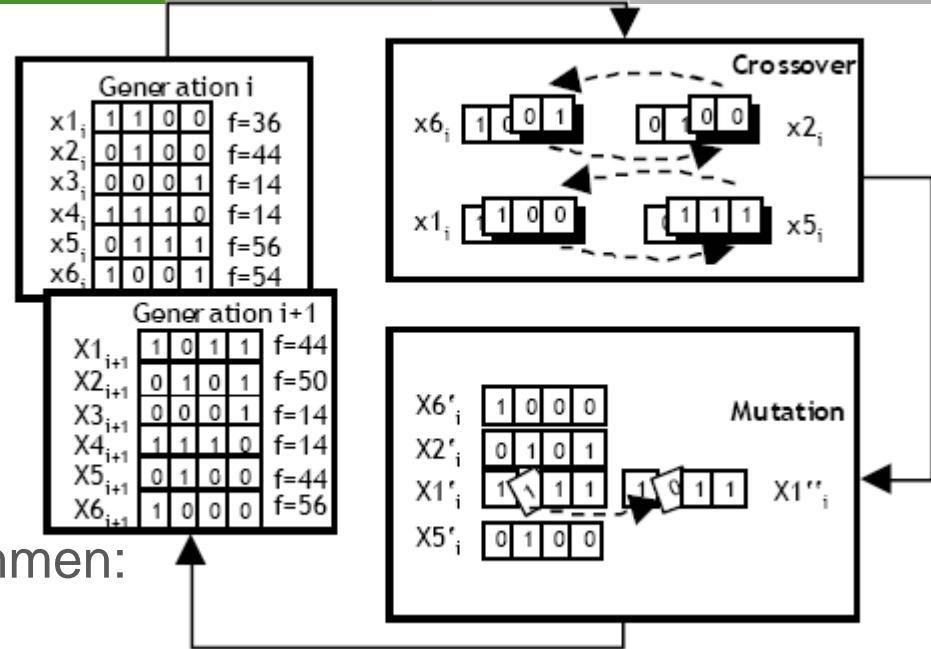
1000|111101 1001|111101
 - N-point crossover
- Mutation:
 - Durch Mutation wird ein Aspekt eines Kandidaten in zufälliger Weise verändert
 - Hierdurch soll verhindert werden, dass der Algorithmus in lokalen Extrema stecken bleibt
 - Realisiert z.B. durch lokales Umklappen eines Bits im Bitstring

Ein einfaches Beispiel

- Problem:
 - Finde maximalen Wert der Funktion $(15x - x^2)$, wobei x zwischen 0 und 15 variieren soll
- Binärkodierung der Chromosomen:
1: 0001, 2: 0010, 3: 0011, ..., 14: 1110, 15: 1111
- Fitnessfunktion: $f(x) = (15x - x^2)$
- Annahmen:
 - Größe der Population: 6
 - Kreuzungswahrscheinlichkeit: $p_c = 0.7$
 - Mutationswahrscheinlichkeit: $p_m = 0.001$
 - Zufällig generierte Population von Chromosomen durch zufälliges Füllen von 6 4-Bitstrings mit Einsen und Nullen
 - Selektion der 4 besten Chromosomen für die nächste Generation

Label	Bitstring	Kodierte Zahl	Fitness	Fitness Ratio, %
x1	1100	12	36	16.5
x2	0100	4	44	20.2
x3	0001	1	14	6.4
x4	1110	14	14	6.4
x5	0111	7	56	25.7
x6	1001	9	54	24.8

Beispiel für einen Zyklus



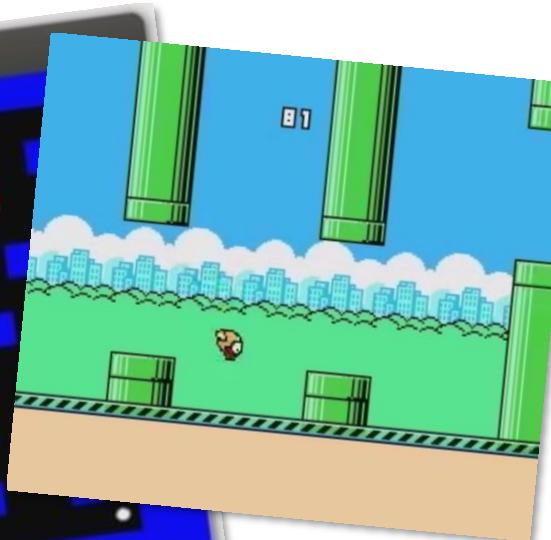
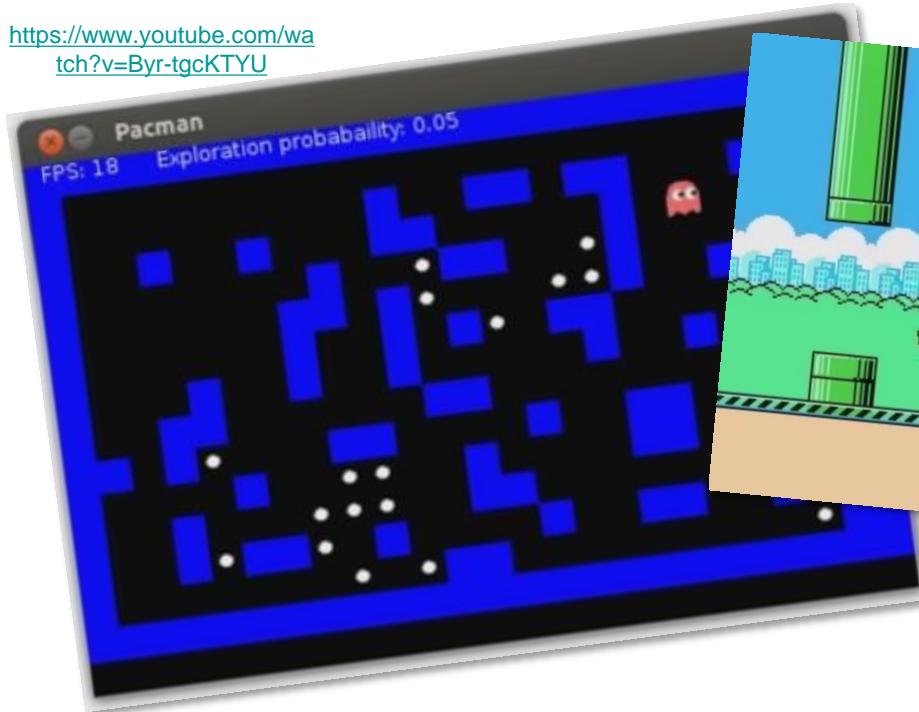
- Vorteile von genetischen Algorithmen:
 - Robust
 - Bewältigen große, komplexe Suchräume
- Nachteile:
 - Benötigen viele Ressourcen
 - Einstellen der Parameter schwierig

<http://brainz.org/15-real-world-applications-genetic-algorithms/>

Reinforcement Learning

Bestärkendes Lernen

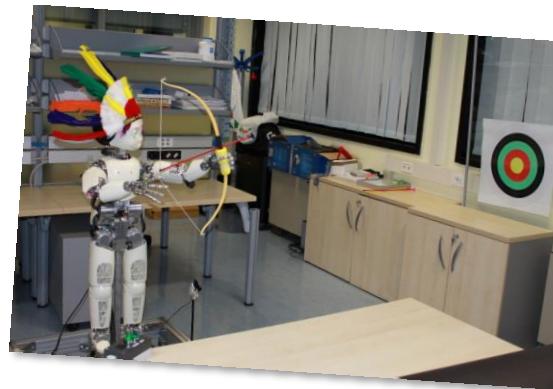
<https://www.youtube.com/watch?v=Byr-tgcKTYU>



<http://cs.stanford.edu/people/ang/?portfolio=autonomous-helicopter-flight>



<https://www.youtube.com/watch?v=xM62SpKAZHU>



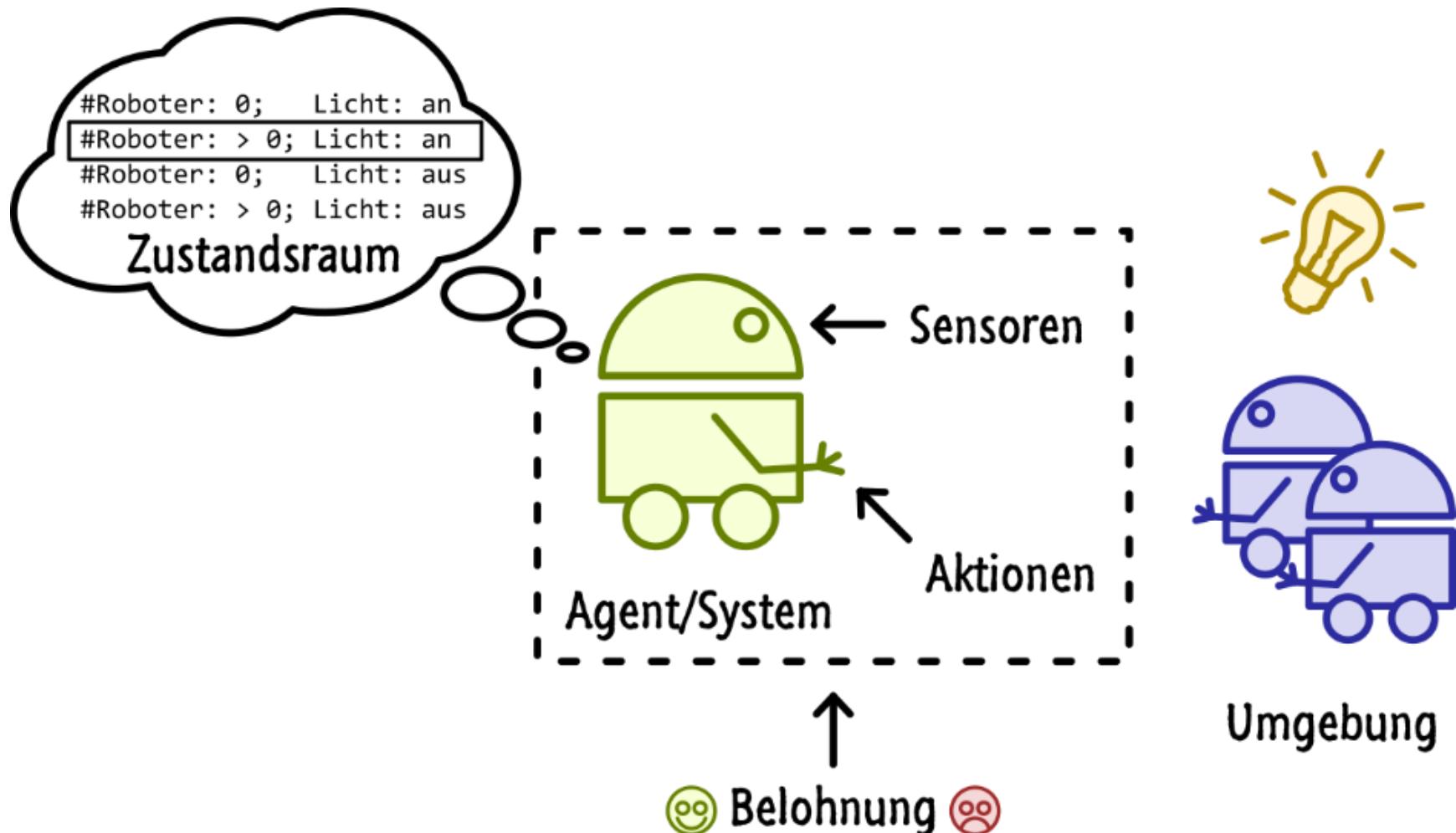
<http://www.mdpi.com/2218-6581/2/3/122/pdf>

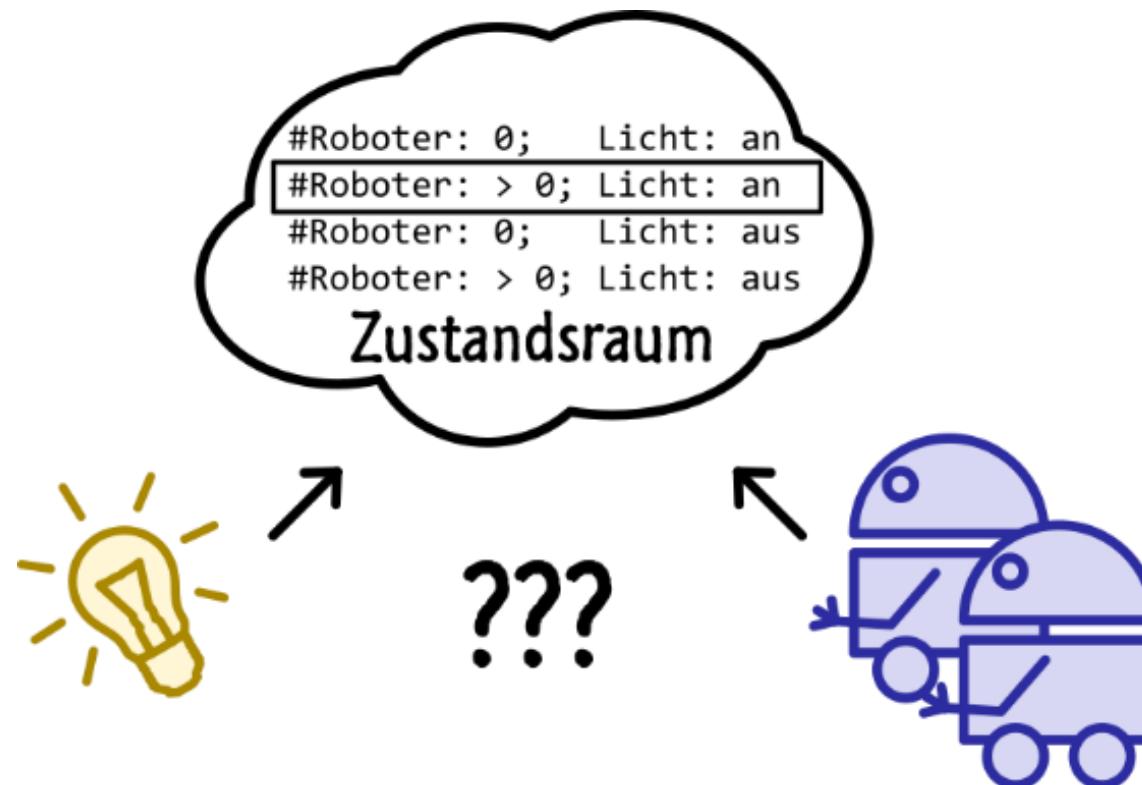
Das System startet ohne Vorwissen, aber lernt mit der Zeit eine (optimale) Lösung.

- **Kein Lehrer, aber sensomotorische Verbindung zur Umgebung**
- Lernen von Informationen über **Ursache und Wirkung, Konsequenzen**
- Lernen der **benötigten Schritte zum Erreichen des Ziels**



Lernen durch Interaktion



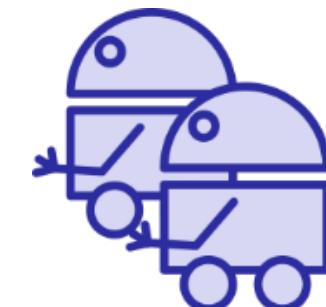


Einfaches Beispiel

- Beschränkung auf **einzelne**, relevante **Merkmale** des Problems
 - Jedes Merkmal hat **unterschiedliche Ausprägungen** (z. B. Licht an/aus)
 - **Merkmale spannen Zustandsraum auf**
- Beispiel: Licht soll nur brennen, wenn weitere Roboter im Raum sind
- Relevante Merkmale
 - Anzahl der Roboter: 0, > 0
 - Licht: an, aus
- Durch Kombination aller Merkmalsausprägungen ergeben sich vier Zustände:

Zustand	#Roboter	Licht
1	0	An
2	0	Aus
3	> 0	An
4	> 0	Aus

Zustandsraum

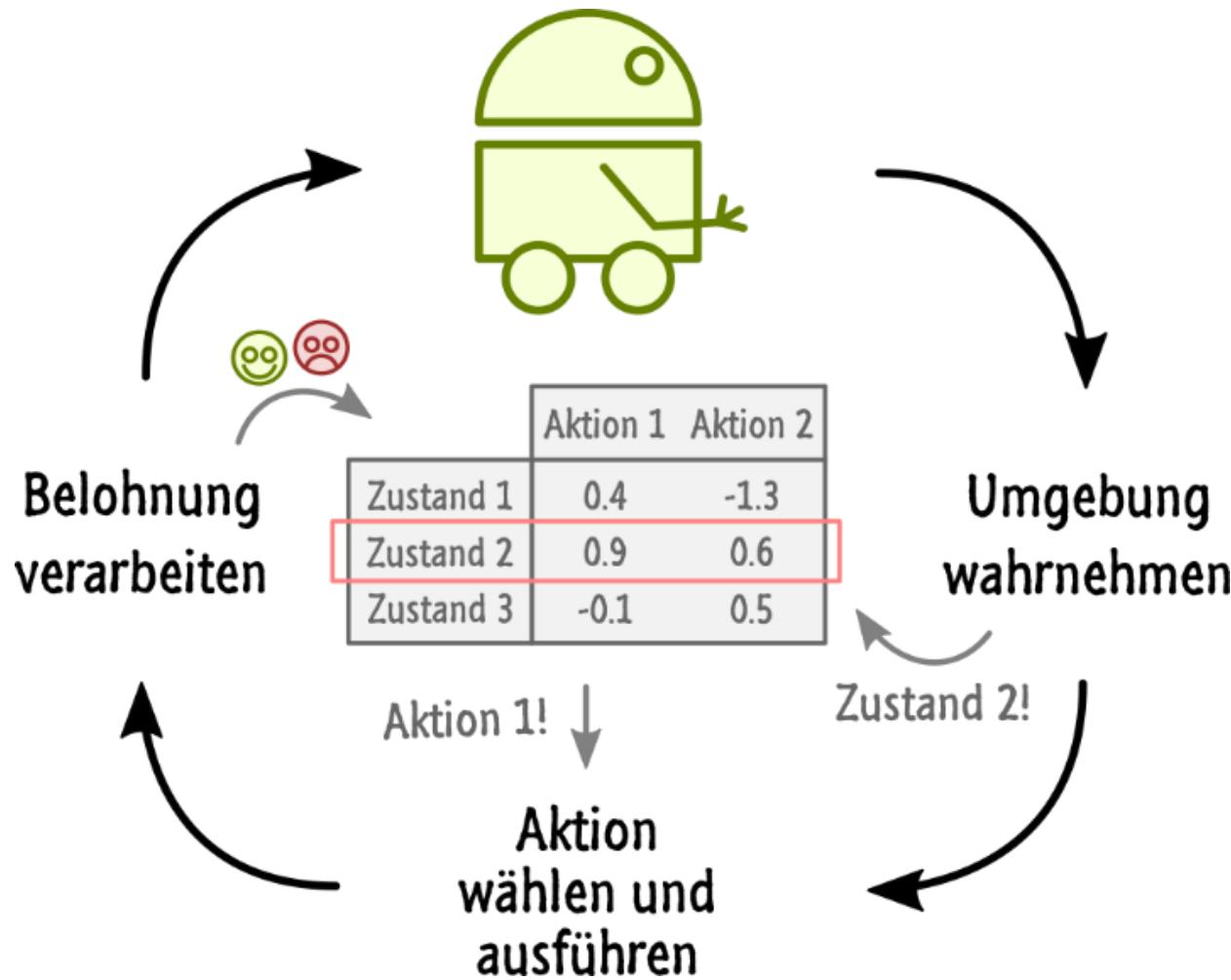


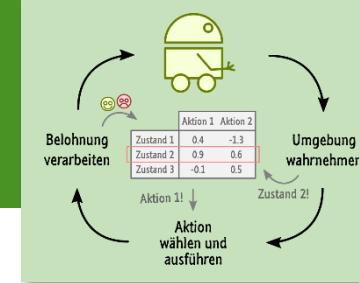
Q-Lernen

Q-Lernen ist nur einer von vielen Ansätzen, wie Bestärkendes Lernen implementiert/realisiert werden kann.



Infinite Mario AI Reinforcement Learning Q-Learning

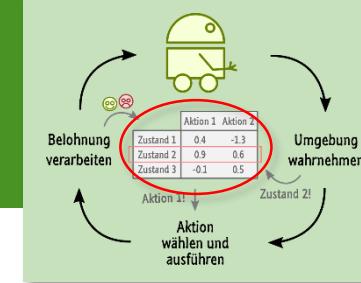




- **Q-Werte-Tabelle**

- **Repräsentiert gesamtes “Wissen” des Systems**
- Zentrale Frage beim Bestärkenden Lernen: Wann ist welche Aktion zielführend?
- Konkret: **In welchem Zustand soll welche Aktion ausgeführt werden?**
- **Werte basieren auf bisher gesammelten Belohnungen**

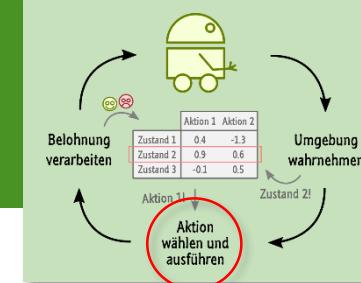
Q-Werte-Tabelle



- **Abbildung von Zuständen und Aktionen auf numerische Werte**
 - Positiver Wert: Aktion scheint zielführend zu sein
 - Negativer Wert: Aktion scheint nicht zielführend zu sein
 - Einschätzung, **wie „gut“ oder „schlecht“** das Ausführen einer Aktion in entsprechendem Zustand ist
- **Initialisierung**
 - Möglichkeit, **Vorwissen** bereit zu stellen
 - Nur **0-Werte: kein Vorwissen**

Zustandsraum

	Aktion 1	Aktion 2	Aktion 3	...
Zustand 1	0.2	-0.4	1.1	...
Zustand 2	-0.3	0.7	0.7	...
Zustand 3	1.3	-0.8	0.9	...
...



- Aktueller Zustand bekannt, aber **welche Aktion soll ausgeführt werden?**
- Ausbeutung vs. Erforschung
 - **Ausbeutung:** Wähle **vermeintlich „beste“ Aktion**
 - Problem: Es könnte *noch* bessere Lösung geben
 - Problem: Nur, weil Aktion ein/mehrere Mal(e) gut/schlecht war, muss dies nicht immer der Fall sein!
 - Gefahr: Tatsächlich optimales Verhalten wird nie gelernt
 - **Erforschung:** Wähle **noch nicht ausgeführte/vermeintlich nicht optimale Aktion**
 - „Wertigkeit“ der vermeintlich optimalen Aktion kann sich geändert haben
 - Nur dadurch ist Lernen des tatsächlich optimalen Verhaltens möglich
 - **Herausforderung: guter Kompromiss**



Jede Aktion muss sehr oft ausprobiert werden, um Belohnung verlässlich anzunähern

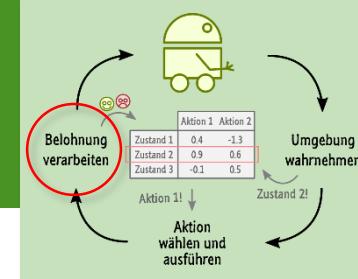




- $0 \leq \epsilon \leq 1$
- Üblicherweise kleines ϵ
 - Ausbeutung: In der Regel "**gierig**" (Wahl der Aktion, die die größte Belohnung verspricht (Suche Maximum in Zeile der Q-Wert Tabelle))
 - Erforschung: Mit **Wahrscheinlichkeit ϵ** Wahl einer **zufälligen Aktion**
- ϵ bestimmt Verhältnis Erforschung/Ausbeutung

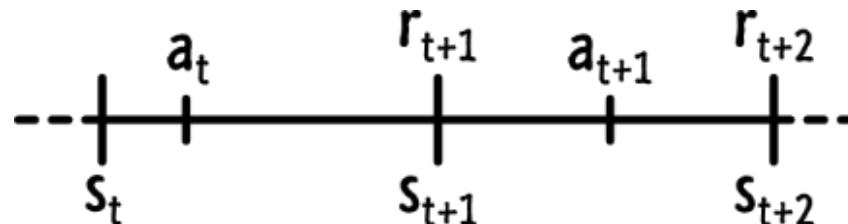
	Aktion 1	Aktion 2	Aktion 3
Zustand 1	0.2	-0.4	1.1
Zustand 2	-0.3	0.7	0.7
Zustand 3	1.3	-0.8	0.9
...	\$\$\$

A yellow arrow points from the bottom left towards the value "1.3" in the third row and first column of the table. The word "\$\$\$" is written vertically along the arrow.

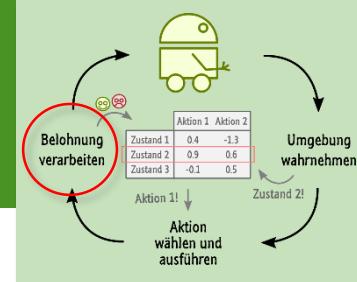


Definitionen

- \mathcal{S} : Menge aller Zustände
- $s_t \in \mathcal{S}$: Zustand
- $\mathcal{A}(s_t)$: Menge aller Aktionen in Zustand s_t
- $a_t \in \mathcal{A}(s_t)$: Aktion im Zustand s_t
- $r_{t+1} \in \mathbb{R}$: **Belohnung** für Aktion a_t im Zustand s_t
- $Q(s_t, a_t)$: **Q-Wert-Funktion**; Q-Wert bei Ausführen von Aktion a_t in Zustand s_t



Aktualisierung



- Agent erhält nach jeder Aktion Belohnung/Bestrafung
- **Aktualisierung** entsprechender Tabellenzelle:
Verrechnung der erhaltenen **Belohnung mit bisherigem Q-Wert**

Der Agent hat keinen Einfluss auf die Belohnung!



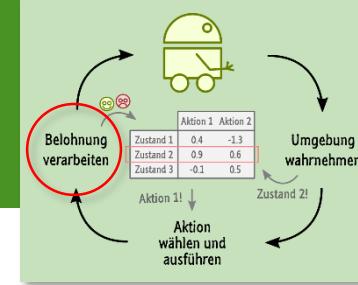
Aktualisierung

- Suche im Nachfolgezustand s_{t+1} optimale Aktion a
- a maximiert Belohnung (größter Q-Wert) im nächsten Schritt
- Algorithmus rechnet mit größtmöglicher zukünftiger Belohnung!
- Aber: Maximale Belohnung muss nicht zwingend eintreten!
- Nachteil: Verrechnung der erwarteten optimalen, nicht der im nächsten Zustand tatsächlich eintretenden Belohnung!

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$



- **Suche im Nachfolgezustand s_{t+1} optimale Aktion a**
- **a maximiert Belohnung (größter Q-Wert) im nächsten Schritt**
- Algorithmus rechnet mit größtmöglicher zukünftiger Belohnung!
- Aber: Maximale Belohnung **muss nicht zwingend eintreten!**
- **Nachteil: Verrechnung der erwarteten optimalen, nicht der im nächsten Zustand tatsächlich eintretenden Belohnung!**



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

- α : **Lern-Rate** $0 < \alpha \leq 1$
 - Beeinflusst **Lerngeschwindigkeit**
 - **Je größer, desto schnelleres Lernen** (neuer Wert hat mehr Einfluss)
- γ : **Diskont-Wert** $0 \leq \gamma \leq 1$
 - Grundidee: **Je früher das Problem gelöst wird, desto besser**
 - Wenn $\gamma \neq 1$ werden **spätere Aktionen unattraktiver** (Diskont der max. zu erwartenden Belohnung)

Gegeben

- $s_t = s_1, s_{t+1} = s_3, a_t = a_2$
- $r_{t+1} = 1.2, \alpha = 0.8, \gamma = 0.9$

	a_1	a_2	a_3
s_1	2.4	-0.4	0.4
s_2	1.3	-1.3	2.2
s_3	-0.5	1.8	2.5
...

Gesucht

- $Q(s_t, a_t) = ?$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Gegeben

- $s_t = s_1, s_{t+1} = s_3, a_t = a_2$
- $r_{t+1} = 1.2, \alpha = 0.8, \gamma = 0.9$

	a_1	a_2	a_3
s_1	2.4	-0.4	0.4
s_2	1.3	-1.3	2.2
s_3	-0.5	1.8	2.5
...

Lösung

- $\max_{a'} Q(s', a') = 2.5$
- $Q(s_t, a_t) = -0.4 + 0.8 \cdot [1.2 + 0.9 \cdot 2.5 - (-0.4)] = 2.68$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Algorithmus

Initialisiere $Q(s, a)$



Wiederhole (für jede Lernepisode)

Initialisiere s

Wiederhole (für jeden Schritt der Episode)

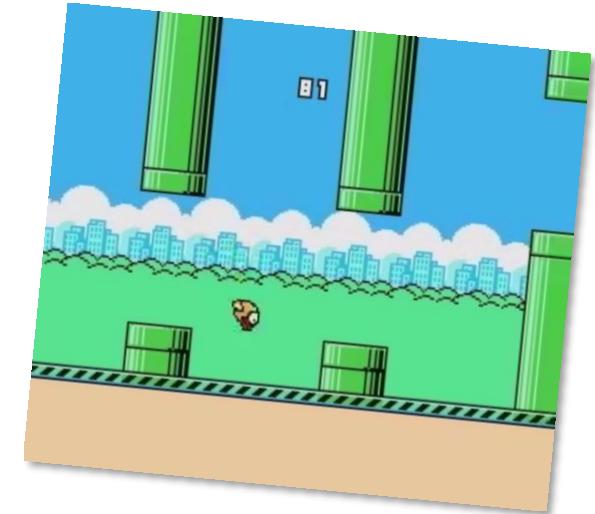
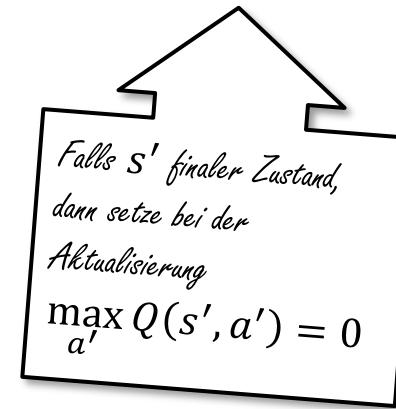
Strategie wählt a auf Basis von s

Führe a aus, beobachte r und s'

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

$s \leftarrow s'$

bis s finaler Zustand





DQN = Deep Q-Learning als Verbindung von Reinforcement Learning und neuronalen Netzen.



- NN ist von Parametern ϕ abhängige Funktion $f(\phi)$
- Grundlegende Idee: Minimierung einer Fehlerfunktion, die den Abstand zur *Grundwahrheit* y misst, zum Beispiel *mean squared error*

$$L(\phi) = (y - f(\phi))^2$$

- Dazu: (Stochastic) Gradient Descent

$$\phi_{i+1} = \phi_i - \alpha \nabla L(\phi_i)$$



- Verwenden Neuronales Netz mit Parametern ϕ um Aktionen auszuwählen

$$Q(s, a, \phi)$$

- wollen theoretisch optimale Strategie als Grundwahrheit verwenden

$$L(\phi) = (Q^*(s, a) - Q(s, a, \phi))^2$$



- Verwenden Neuronales Netz mit Parametern ϕ um Aktionen auszuwählen

$$Q(s, a, \phi)$$

- wollen theoretisch optimale Strategie als Grundwahrheit verwenden

$$L(\phi) = (Q^*(s, a) - Q(s, a, \phi))^2$$

- ▶ Lösung: Verwenden bisher gelernten Q-Wert als Grundwahrheit y

$$L(\theta_i)$$

$$= \left((r(s_t, a_t, s_{t+1}) + \gamma \max_{a' \in A} Q(s_{t+1}, a', \tilde{\theta})) - Q(s_t, a_t; \theta_i) \right)^2$$



Während des DQN Trainings eigentlich 2 Netze mit identischer Struktur

Ziel-Netz:

- liefert "Ground-Truth"
- wird nicht trainiert

Trainings-Netz:

- wählt Aktionen des Agenten (ϵ -greedy)
- wird trainiert

Nach C Schritten werden die Gewichte des Ziel-Netzes durch die gelernten Gewichte des Trainings-Netzes ersetzt.
Durch diese Architektur wird das Training "stabiler" und divergiert nicht so einfach.



Experience Replay:

Während des Trainings werden die durchgeföhrten Übergänge in einer Erinnerungsmenge gespeichert und in jedem Lernschritt wird ein Batch zufälliger Übergänge aus dieser Menge verwendet.

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Quelle: <https://www.nature.com/articles/nature14236/>