Opponent Modeling

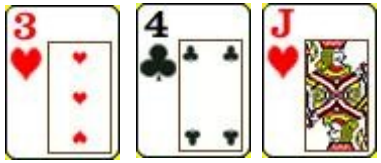
- Einführung
- Loki & Poki
  - Effektive Handstärke
  - Opponent Modeling in Loki
  - Erweiterungen in Poki
  - Experimentelle Ergebnisse
- Vexbot
  - Anwendungsbeispiel
  - Schwierigkeiten
  - Lösungsansätze
- Fazit

Einführung

- Poker ist ein Spiel mit unvollständigen Informationen
- Die optimale Aktion, ist die Aktion die man auch machen würde, wenn man die gegnerischen Karten kennt
- Man versucht aufgrund der Spielweise des Gegners Rückschlüsse auf seine Hand zu ziehen
- Die eigene Spielweise hängt stark von der Strategie der Gegner ab

Effektive Handstärke

Eigene Karten: 

Flop: 

Der Gegner kann 1081 verschiedene Kartenkombinationen auf der Hand halten.

Handstärke (HS_n)

444 Kombinationen sind besser
9 Kombinationen sind gleich gut
628 Kombinationen sind schlechter

-> Handstärke = 0.585

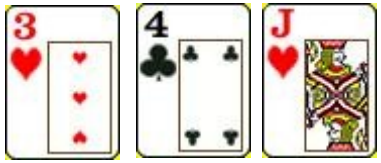
Potential der Hand (P_{pot})

91.981 Möglichkeiten: Am Flop hinten -> Am River vorne
1.036 Möglichkeiten: Am Flop hinten -> Am River gleich auf
346.543 Möglichkeiten: Am Flop hinten -> Am River hinten

-> Potential = 0.209

Effektive Handstärke

Eigene Karten: 

Flop: 

Die effektive Handstärke (EHS) ist die Summe der Wahrscheinlichkeit, dass man momentan die besten Hand hat und der Wahrscheinlichkeit, dass man momentan hinten, aber am Ende vorne ist.

$$EHS = HS_n + (1 - HS_n) * P_{pot}$$

Opponent Modeling in Loki

- Von einer zufälligen Hand beim Gegner auszugehen, ist keine realistische Vermutung
- Loki speichert für jeden Gegner eine Tabelle mit Gewichten für jede mögliche Hand.
- Die Gewichte ändern sich während einer Hand aufgrund der Aktionen des Gegners
- Zwei Probleme:
  - Wie bestimmt man die initialen Gewichte?
  - Wie passt man die Gewichte aufgrund der Aktionen des Gegners an?

Opponent Modeling in Loki

Bestimmung der initialen Gewichte:

- Es gibt 1326 Kombinationen von Startkarten aber nur 169 Typen von Starthänden
- Für jede dieser Handtypen wurde eine Simulation eines Pokerspiels mit jeweils 1.000.000 Spielen gegen 9 zufällige Hände durchgeführt
- Daraus wurde die *income rate* für jeden Typ bestimmt
- Die Bestimmung der initialen Gewichte für die Karten eines Gegners wird anhand seines Pre Flop-Verhaltens vorgenommen

Opponent Modeling in Loki

Beispiel:

- Angenommen ein Gegner geht in 30% der Fälle vor dem Flop mit
- Das bedeutet er called im Durchschnitt mit Karten die durchschnittlich 0.2 Bets pro Hand gewinnen
- Eine angenommene Varianz führt zu einer Schwankung von 0.1 Bets
- Erwarteter Gewinn der Startkarten:
  - < 0.1 Bets -> Gewicht = 0.01
  - > 0.3 Bets-> Gewicht = 1
  - 0.1 Bets < Gewinn < 0.3 Bets -> Gewicht wird linear interpoliert

Opponent Modeling in Loki

Anpassung der Gewichte:

- Zur Anpassung der Gewichte während einer Hand, werden wiederum die Häufigkeiten der Aktionen des Gegners beobachtet
- Loki unterscheidet 9 verscheidene Situationen in denen ein Gegner folden, checken/callen oder betten/raisen kann
- Unterschieden wird nach:
  - Der Setzrunde (Flop, Turn, River)
  - Den Bets die der Gegner aufbringen müsste um mitzugehen (0, 1, >1)

Opponent Modeling in Loki

Anpassung der Gewichte (Beispielsituationen):

- Situation 1:

  Wir sind am Flop als erster dran und setzen:

  -> Situation des Gegners:

  Setzrunde: Flop

  Anzahl der Bets um zu callen: 1

- Situation 2:

  Wir sind am River als erster dran und checken:

  ->Situation des Gegners:

  Setzrunde: River

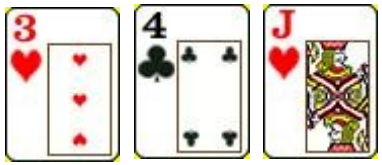
  Anzahl der Bets um zu callen: 0

Opponent Modeling in Loki

Anpassung der Gewichte:

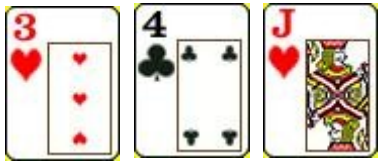
- Aufgrund der beobachteten Häufigkeiten der Aktionen des Gegners werden wieder Rückschlüsse auf seine Kartenstärke gezogen
- Daher werden die Kartenkombinationen mit einem Faktor zwischen 0 und 1 multipliziert um das Gewicht anzupassen

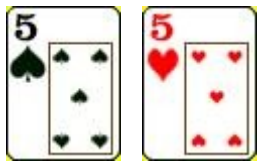
Opponent Modeling in Loki

Beispiel: Flop: 

- Situation:
  - Gegner ist vor dem Flop mitgegangen und setzt jetzt
- Beobachtungen:
  - Die gemachten Beobachtungen deuten darauf hin, dass der Gegner eine Durchschnittliche Handstärke von 0.6 braucht um zu setzen
  - Wir gehen von einer Varianz aus, die eine Schwankung der Handstärke um 0.2 ergibt.

Opponent Modeling in Loki

Beispiel: Flop: 

Hand	Gewicht	EHS	Rwt	Nwt
	0.01	0.99	1.00	0.01
	1.00	0.94	1.00	1.00
	0.70	0.74	0.85	0.60
	0.90	0.22	0.01	0.01

Erweiterungen in Poki

- Poki ist die Weiterentwicklung von Loki
- Erweitert die Situationen in denen die Häufigkeit der Aktionen des Gegners gezählt werden
- Kontext einer Situation
  - Setzrunde (Flop, Turn, River)
  - Anzahl der Bets um mitzugehen (0,1,>1)
  - Bets die der Gegner zuletzt gecalled hat (0, >1)
  - Letzte Aktion des Gegners (Bet/Raise, kein Bet/Raise)

Experimentelle Ergebnisse

- Poki wurde auf IRC Poker Channels getestet
  - Channel für Einsteiger (#holdem1)
  - Channel für Fortgeschrittene (#holdem2)
  - Gespielt wurden über 20.000 Hände
- Gewinnrate:
  - #holdem1: +0.22 sb/h
  - #holdem2: + 0.8 sb/h

Vexbot

- Speziell für 2 Spieler Matches erstellt
- Suchbaumbasiertes vorgehen: Repräsentation aller denkbaren Spielverläufe in einer Baumstruktur
- Berechnet die erwarteten Gewinne von fold, check/call und bet/raise
- Auswahl der Aktion nach Miximax- oder Miximix-Strategie

Vexbot

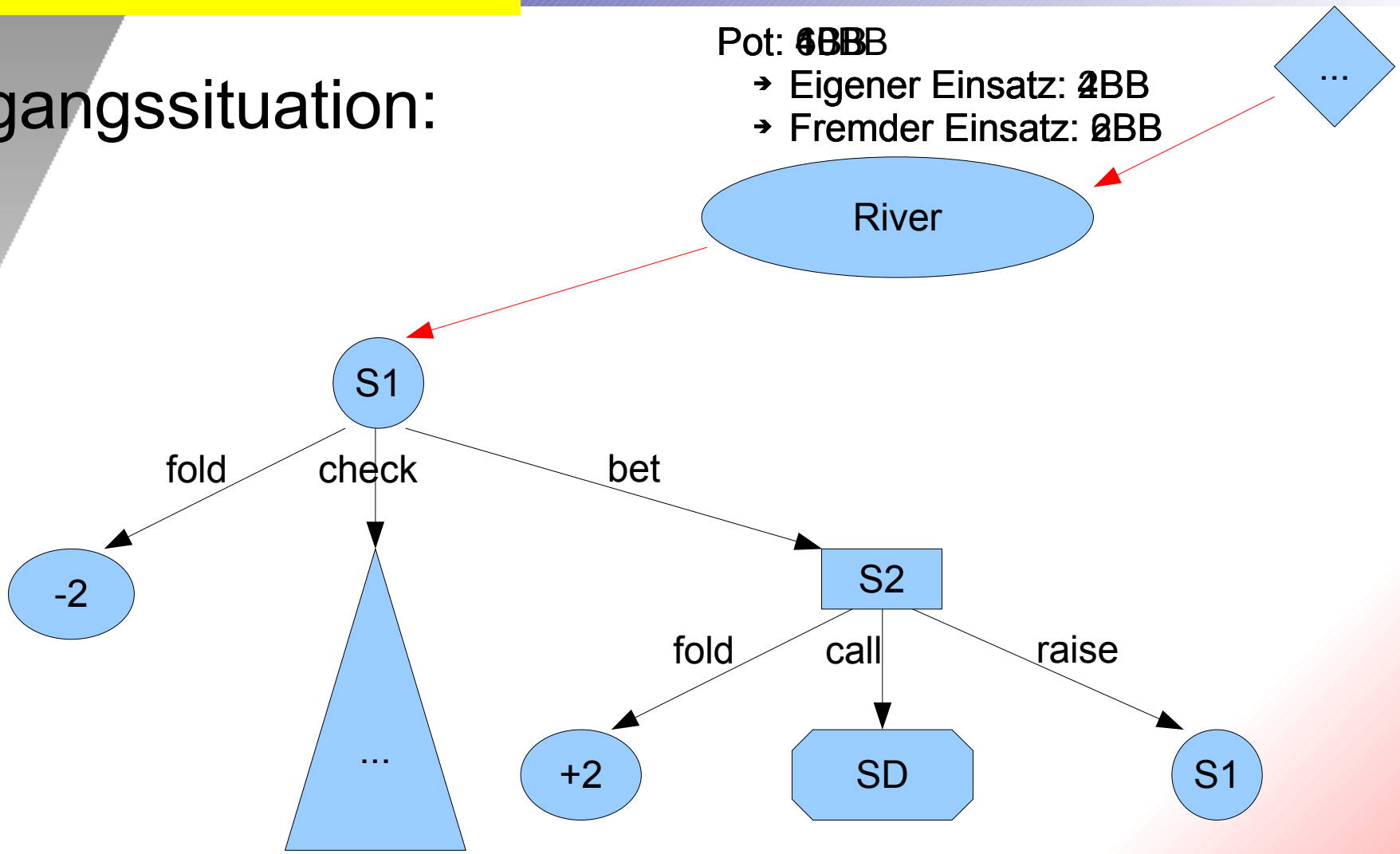
- Elemente des Suchbaums:
  - Decision Node
  - Opponent Decision Node
  - Chance Node
  - Leaf Node
- Opponent Modeling durch Speicherung der beobachteten Spielverläufe / „bet-sequences“
  - Häufigkeit von bestimmten Gegnerentscheidungen
  - Histogramme geben Auskunft über erwartete Handstärke

Vexbot - Anwendungsbeispiel

- Situation:
  - Der Pot enthält 4 Big Blinds bis zum River
  - Nach dem River betten wir und der Gegner reagiert mit einem raise seinerseits
  - Eigene Handstärke: 0.7
- Was jetzt? Fold, Call oder Raise?

Vexbot - Anwendungsbeispiel

Ausgangssituation:

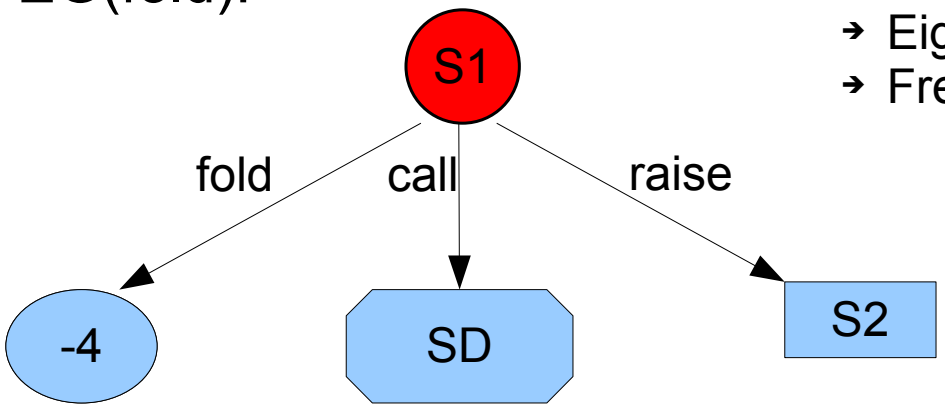


Vexbot - Anwendungsbeispiel

Was jetzt ?

- Berechnung der erwarteten Gewinne (EG) von: fold, call und raise

1) EG(fold):



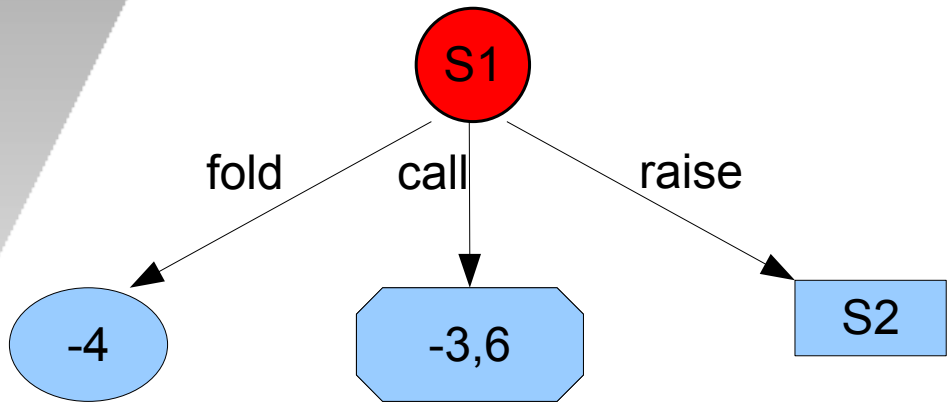
Pot: 10BB
- Eigener Einsatz: 4BB
- Fremder Einsatz: 6BB

EG(fold) = -4 BB

Vexbot - Anwendungsbeispiel

2) EG(call):

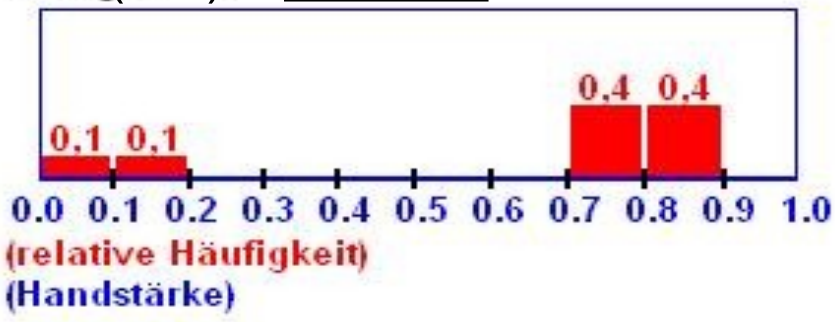
Pot: 12BB
- Eigener Einsatz: 6BB
- Fremder Einsatz: 6BB



EG(call) = p(win) * Gewinn – p(loose) * Verlust
EG(call) = p(win) * 6 BB – p(loose) * 6 BB
EG(call) = 0,2 * 6 BB – 0,8 * 6 BB
EG(call) = 1,2 BB – 4,8 BB
EG(call) = – 3,6 BB

Wie groß sind die eigenen Gewinnchancen?

Histogramm:

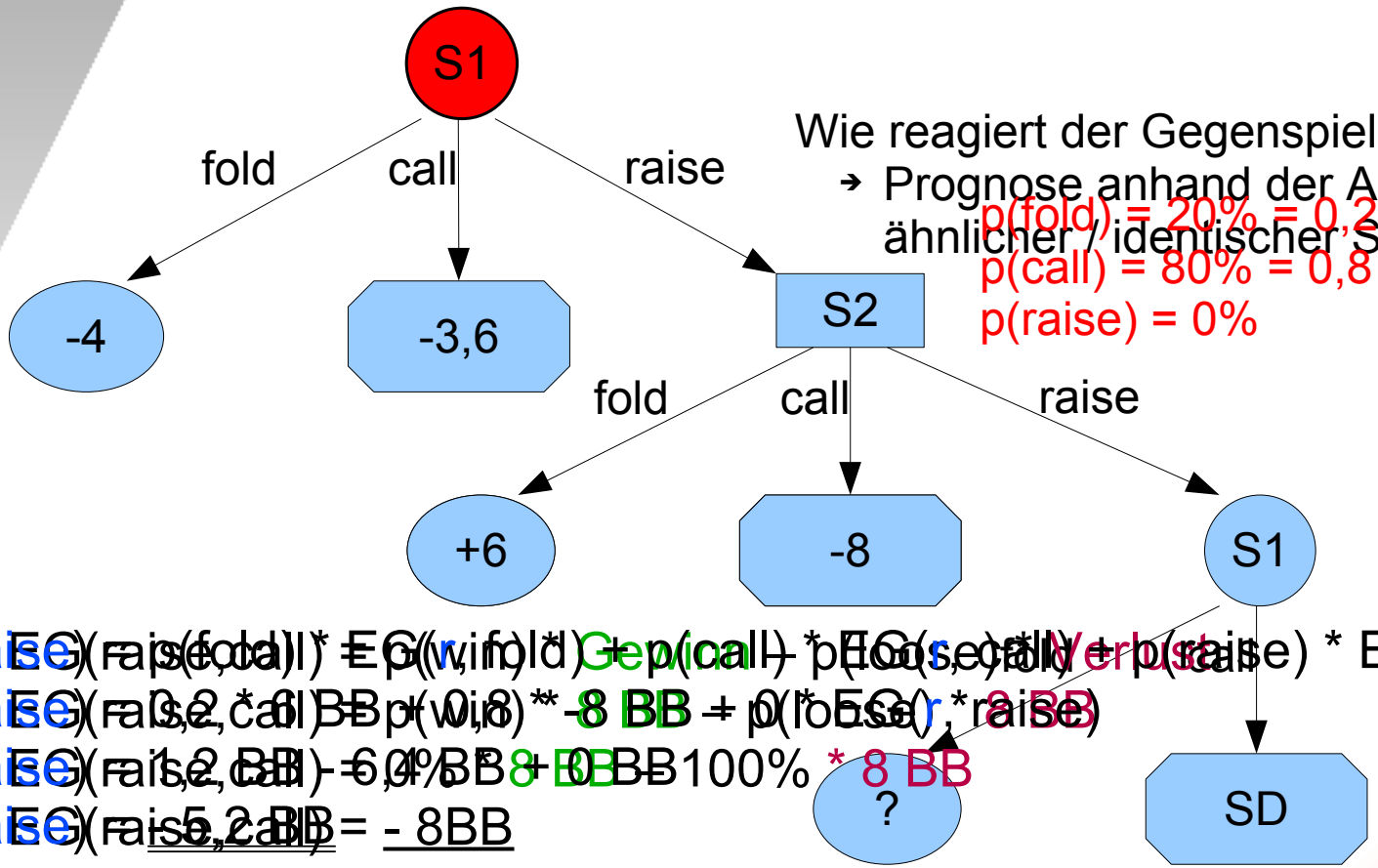


Zur Erinnerung: eigene Handstärke = 0,7
- p(win) = 20% = 0,2
- p(loose) = 80% = 0,8

Vexbot - Anwendungsbeispiel

3) EG(raise):

Pot: 10BB
- Eigener Einsatz: 8BB
- Fremder Einsatz: 6BB



EG(raise) = p(fold) * EG(r, fold) + p(call) * EG(r, call) + p(raise) * EG(r, raise)

EG(raise) = 0,2 * 6BB + 0,8 * -8 BB + 0 * EG(r, raise)

EG(raise) = 1,2 BB - 6,4 BB + 0 BB

EG(raise) = - 5,2 BB

Gewinn

Verlust

100% * 8 BB

= - 8BB

Vexbot - Anwendungsbeispiel

Ergebnis:

EG(fold) = -4 BB

EG(call) = -3,6 BB

EG(raise) = -5,2 BB

Entscheidung nach Miximax-Strategie => **CALL**

Vexbot - Schwierigkeiten

- Lernprozess muss schnell erfolgen, da Matches in der Regel aus wenigen Runden bestehen (oft weniger als 100 Runden)
- Erfahrene Pokerspieler wechseln häufig ihr Spielverhalten um möglichst unberechenbar zu bleiben
- Unvollständige Informationen erschweren Beurteilung des Gegnerverhaltens

Vexbot - Schwierigkeiten

- Anzahl der verschiedenen Spielverläufe weit größer als Anzahl der zu Spielenden Runden
  - Es gibt bei 2 Spieler - Matches allein schon $9^4 = 6561$ verschiedene „bet-sequences“ pro Spieler, die zu einem Showdown führen
- Schnelles Lernverhalten bei Vexbot ein großes Problem, da identische Spielverläufe nur selten auftreten
  - → Häufig für aktuelle Situation keine oder nur wenige Daten verfügbar

Vexbot - Lösungsansätze

- Abstraktionsmodell
  - Gruppierung bzw. Gleichbehandlung von „bet-sequences“ mit gemeinsamen Eigenschaften
    - ermöglicht es Aussagen für bisher noch nie oder selten aufgetretene Situationen zu treffen
    - z.B.: alle „bet-sequences“ mit der gleichen Anzahl an raises unabhängig der genauen Setzreihenfolge
  - Mehrere Abstraktionsschichten notwendig, um verfügbare Daten möglichst gut zu nutzen
    - Von sehr starker Abstraktion bis gar keiner Abstraktion
    - Trade-Off zwischen größerer Relevanz und größerer Anzahl verwertbarer Beobachtungen

Vexbot - Lösungsansätze

- Wahl der Abstraktionsschicht
  - Intuitiver Ansatz: Verwendung einer Schicht mit möglichst großer Aussagekraft bei gegebener Anzahl an Daten
  - Bei Vexbot: Verwendung aller Schichten und Gewichtung der Ergebnisse mit Anzahl n der dort verfügbaren Beobachtungen:
    - → Gewichtung: $(1-m^n)$ des noch verbliebenen Gewichts

Vexbot - Lösungsansätze

- Wahl der Abstraktionsschicht - Beispiel
  - Annahmen:
    - 3 Schichten: A0 (keine Abstraktion), A1 und A2 (starke Abstraktion)
    - Anzahl Beobachtungen: A0 = 5; A1 = 20; A2 = 75
    - m = 0.95
  - Gewichtungen:
    - A0: $(1-m^5)=0{,}23$
    - A1: $(1-m^{20})=0{,}64 \longrightarrow 0{,}64*(1-0{,}23)\approx 0{,}5$
    - A2: $(1-m^{75})=0{,}98 \longrightarrow 0{,}98*(1-0{,}23-0{,}5)\approx 0{,}27$

Fazit

- Viele verschiedene Ansätze mit jeweiligen Stärken & Schwächen
  - Suchbaumbasierte Ansätze zeigen die besten Ergebnisse, sind bisher jedoch nur für maximal 2 Spieler Matches realisierbar
- Als einzige Richtlinie hat sich herausgestellt, dass Programme mit weniger statischen Annahmen meist besser sind

Vielen Dank!

Noch Fragen

Lars Meyer - Thomas Görge