General Game Playing

Martin Günther

mguenthe@uos.de

17. Juni 2010

1997: Deep Blue schlägt Kasparov

Warum General Game Playing? (1)

General Game Player sind Systeme, die in der Lage sind

- eine formale Beschreibung beliebiger Spiele zu verarbeiten
- auf Grundlage dieser Beschreibung effektiv zu spielen
- $\rightarrow$ auf ein konkretes Spiel spezialisierte Algorithmen unmöglich!

Warum General Game Playing? (1)

General Game Player sind Systeme, die in der Lage sind

- eine formale Beschreibung beliebiger Spiele zu verarbeiten
- auf Grundlage dieser Beschreibung effektiv zu spielen
- $\rightarrow$ auf ein konkretes Spiel spezialisierte Algorithmen unmöglich!

Grundprinzip

Ein Schlüsselindikator für Intelligenz ist die Fähigkeit, sich an völlig unbekannte Umgebungen anzupassen.

Warum General Game Playing? (2)

Relevante Teilgebiete der KI:

- Computer Game Playing
- Handlungsplanung
- automatisches Schließen, Wissensrepräsentation
- Maschinelles Lernen
- Spiel- und Entscheidungstheorie

AAAI General Game Playing Competition

- seit 2005 Wettbewerb auf großer KI-Konferenz (AAAI)
- 10,000 $ Siegprämie
- jeder kann mitmachen

Endliche Spiele mit simultanen Zügen (1)

Endlicher Spiele

- endliche Anzahl von Positionen (= „Zuständen“)
- 1 Anfangszustand, ≥ 1 Endzustände
- jedes Spiel in endlicher Anzahl von Zügen beendet

Endliche Spiele mit simultanen Zügen (1)

Endlicher Spiele

- endliche Anzahl von Positionen (= „Zuständen“)
- 1 Anfangszustand, ≥ 1 Endzustände
- jedes Spiel in endlicher Anzahl von Zügen beendet

Endliche Spieler

- fixe endliche Anzahl Spieler
- endliche Anzahl möglicher Züge (= „Aktionen“)

Endliche Spiele mit simultanen Zügen (2)

vollständige Information

- vollständige Information: Spieler kennen gesamten Zustand
- Zustand ändert sich nur aufgrund der Aktionen der Spieler
- kein Zufall
- aber: simultane Züge erlaubt

Endliche Spiele mit simultanen Zügen (2)

vollständige Information

- vollständige Information: Spieler kennen gesamten Zustand
- Zustand ändert sich nur aufgrund der Aktionen der Spieler
- kein Zufall
- aber: simultane Züge erlaubt

Beispiele:

- Tic-Tac-Toe, Dame, Mühle, Schach, Go, Othello
- n-Damen-Problem, Verschiebespiel, Blockwelt
- Halma
- Schere-Stein-Papier, „Tic-Tic-Toe“

Endliche Spiele mit simultanen Zügen (2)

vollständige Information

- vollständige Information: Spieler kennen gesamten Zustand
- Zustand ändert sich nur aufgrund der Aktionen der Spieler
- kein Zufall
- aber: simultane Züge erlaubt

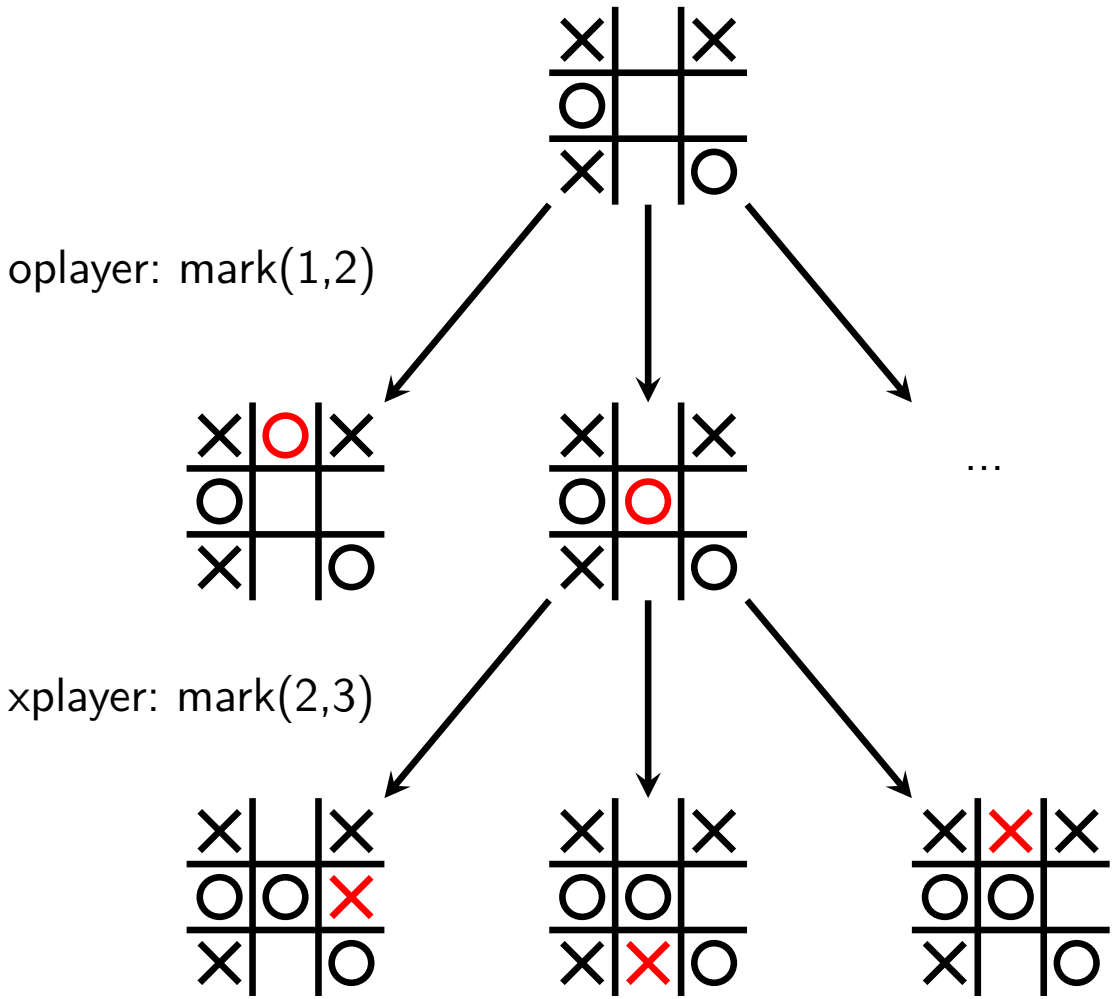
Beispiele:

- Tic-Tac-Toe, Dame, Mühle, Schach, Go, Othello
- n-Damen-Problem, Verschiebespiel, Blockwelt
- Halma
- Schere-Stein-Papier, „Tic-Tic-Toe“

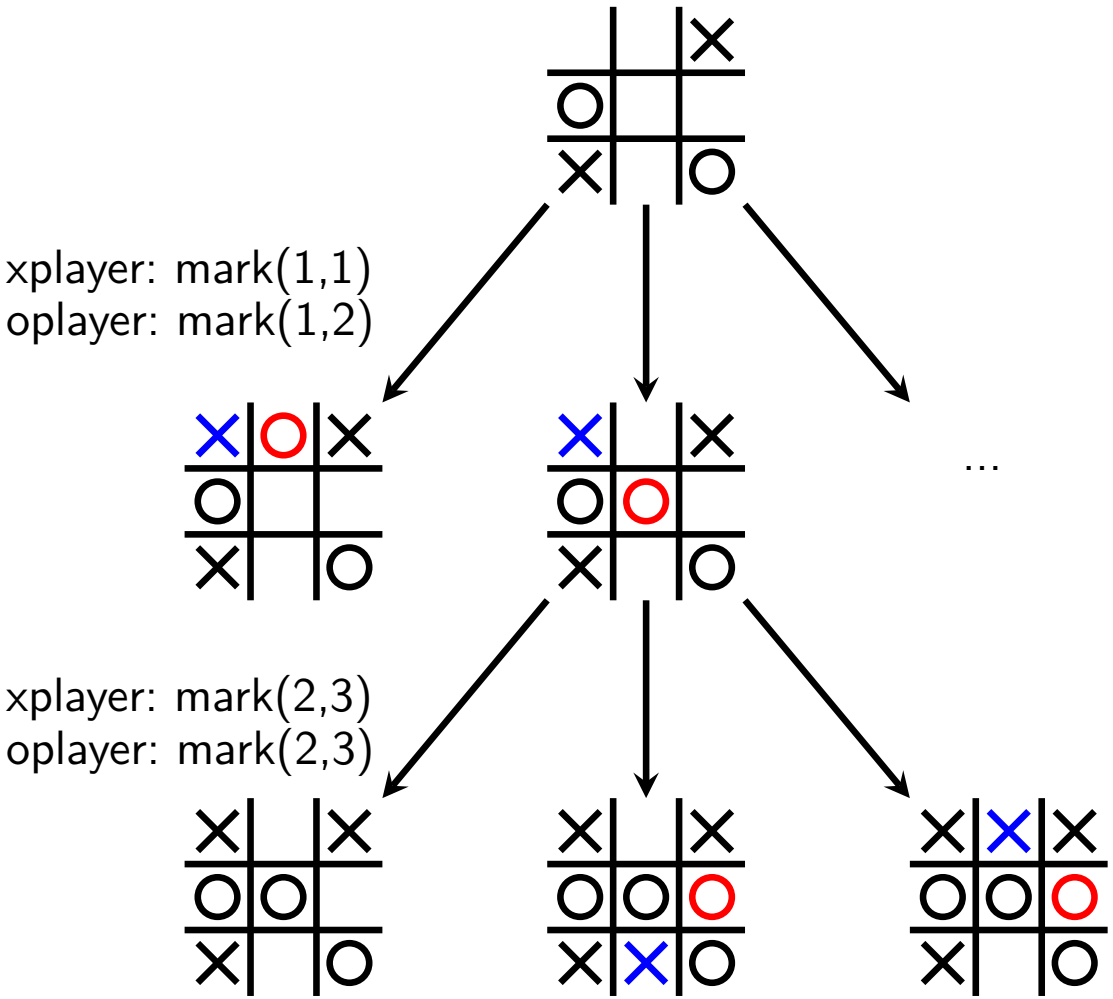
Gegenbeispiele:

- Skat, Poker (noch nicht)
- Mensch ärgere dich nicht, Siedler (noch nicht)
- Star Craft, Call of Duty

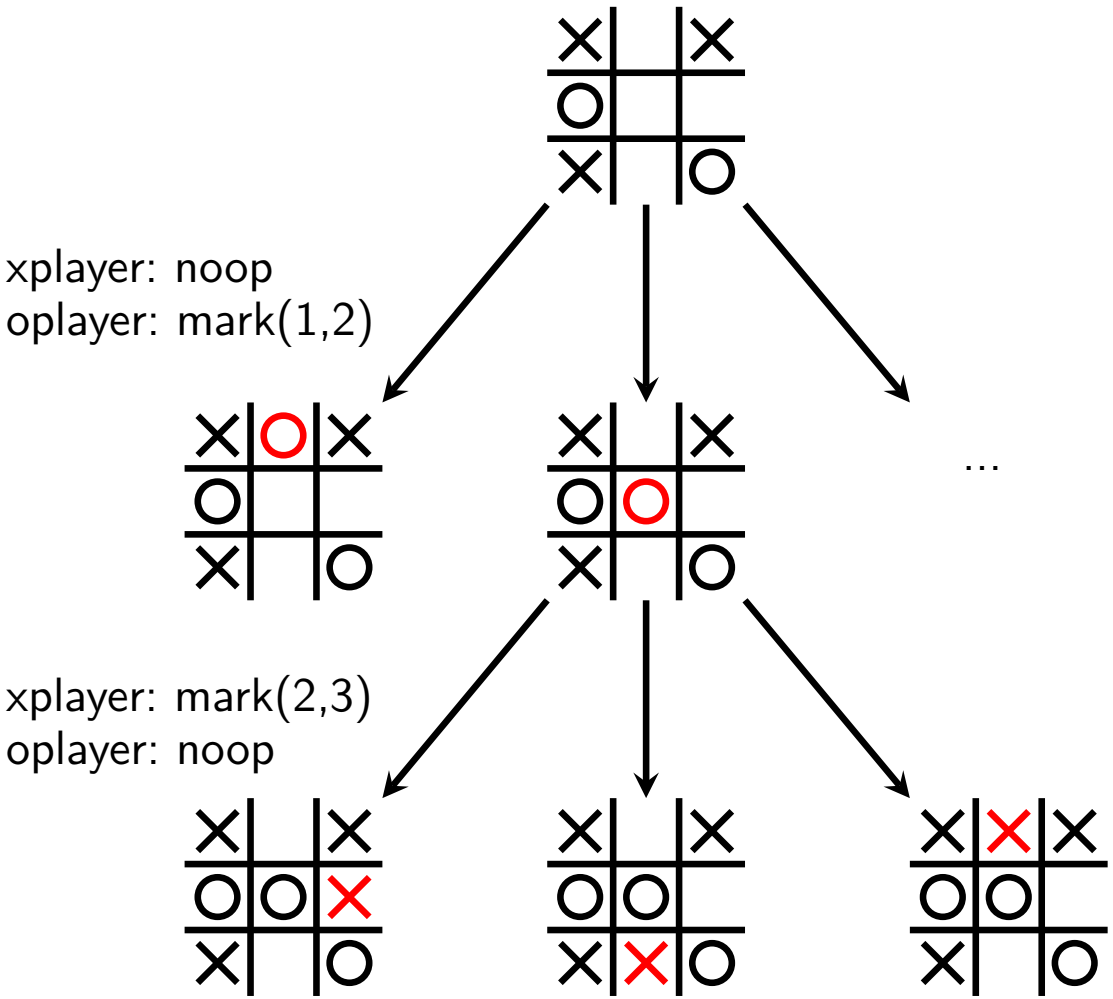
Spielgraphen: Tic-Tac-Toe



Spielgraphen: Tic-Tac-Toe mit simultanen Zügen



Spielgraphen: Tic-Tac-Toe mit No-Ops



Kodierungsmöglichkeiten

Endliche Automaten: astronomisch große Zustandsräume

Kodierungsmöglichkeiten

Endliche Automaten: astronomisch große Zustandsräume

Listen/Tabellen: s.o.

Kodierungsmöglichkeiten

Endliche Automaten: astronomisch große Zustandsräume

Listen/Tabellen: s.o.

prozedurale Programme: eine Möglichkeit: Programm schreiben, das legale Züge und Nachfolgezustände generiert etc. Problem: Analyse der Semantik von prozeduralen Programmen sehr schwer

Kodierungsmöglichkeiten

Endliche Automaten: astronomisch große Zustandsräume

Listen/Tabellen: s.o.

prozedurale Programme: eine Möglichkeit: Programm schreiben, das legale Züge und Nachfolgezustände generiert etc. Problem: Analyse der Semantik von prozeduralen Programmen sehr schwer

Logik: es gibt fertige Interpreter und Compiler; leichter zu analysieren als prozedurale Kodierungen

Tic-Tac-Toe: Vokabular

- **Objekt-Konstanten**

`xplayer, oplayer`	Rollen
`x, o, b`	Markierungen

Tic-Tac-Toe: Vokabular

- **Objekt-Konstanten**

`xplayer, oplayer`	Rollen
`x, o, b`	Markierungen

- **Funktionen**

`cell(number,number,mark)`	Fluent
`control(player)`	Fluent
`mark(number,number)`	Zug

Tic-Tac-Toe: Vokabular

- **Objekt-Konstanten**

`xplayer, oplayer`	Rollen
`x, o, b`	Markierungen

- **Funktionen**

`cell(number,number,mark)`	Fluent
`control(player)`	Fluent
`mark(number,number)`	Zug

- **Relationen**

`row(number,mark)`
`column(number,mark)`
`diagonal(mark)`
`line(mark)`
`open`

Tic-Tac-Toe: Anfangszustand

```
init(cell(1,1,b))
init(cell(1,2,b))
init(cell(1,3,b))
init(cell(2,1,b))
init(cell(2,2,b))
init(cell(2,3,b))
init(cell(3,1,b))
init(cell(3,2,b))
init(cell(3,3,b))
init(control(xplayer))
```

Tic-Tac-Toe: Erlaubte Züge

```
legal(P,mark(X,Y)) <=
    true(cell(X,Y,b)) ∧
    true(control(P))
```

Tic-Tac-Toe: Erlaubte Züge

```
legal(P,mark(X,Y)) <=
    true(cell(X,Y,b)) ∧
    true(control(P))

legal(xplayer,noop) <=
    true(control(oplayer))
legal(oplayer,noop) <=
    true(control(xplayer))
```

Tic-Tac-Toe: Update (1)

```
next(cell(X,Y,x)) <=
    does(xplayer,mark(X,Y))
next(cell(X,Y,o)) <=
    does(oplayer,mark(X,Y))
```

Tic-Tac-Toe: Update (1)

```
next(cell(X,Y,x)) <=
    does(xplayer,mark(X,Y))
next(cell(X,Y,o)) <=
    does(oplayer,mark(X,Y))

next(cell(X1,Y1,Mark)) <=
    true(cell(X1,Y1,Mark)) ∧
    does(P,mark(X2,Y2)) ∧
    ((X1 ≠ X2) ∨ (Y1 ≠ Y2))
```

Tic-Tac-Toe: Update (2)

```
next(control(xplayer)) <=
    true(control(oplayer))
next(control(oplayer)) <=
    true(control(xplayer))
```

Tic-Tac-Toe: Spielende

```
terminal <= line(x) ∨ line(o)
terminal <= ¬open

open <= true(cell(X,Y,b))
```

Tic-Tac-Toe: Spielende

```
terminal <= line(x) ∨ line(o)
terminal <= ¬open

open <= true(cell(X,Y,b))

line(Mark) <= row(X,Mark)
line(Mark) <= column(Y,Mark)
line(Mark) <= diagonal(Mark)
```

Tic-Tac-Toe: Hilfsprädikate

```
row(X,Mark) <=
    true(cell(X,1,Mark)) ∧
    true(cell(X,2,Mark)) ∧
    true(cell(X,3,Mark))
column(Y,Mark) <=
    true(cell(1,Y,Mark)) ∧
    true(cell(2,Y,Mark)) ∧
    true(cell(3,Y,Mark))
diagonal(Mark) <=
    true(cell(1,1,Mark)) ∧
    true(cell(2,2,Mark)) ∧
    true(cell(3,3,Mark))
diagonal(Mark) <=
    true(cell(1,3,Mark)) ∧
    true(cell(2,2,Mark)) ∧
    true(cell(3,1,Mark))
```

Tic-Tac-Toe: Gewinnwerte

```
goal(xplayer,100) <= line(x)
goal(xplayer,50)  <= ¬line(x) ∧ ¬line(o) ∧ ¬open
goal(xplayer,0)   <= line(o)

goal(oplayer,100) <= line(o)
goal(oplayer,50)  <= ¬line(x) ∧ ¬line(o) ∧ ¬open
goal(oplayer,0)   <= line(x)
```

HTTP-Kommunikation

Netzwerkmodell:

- der Player ist der „Server“
- der Gamemaster ist der „Client“
- Gamemaster öffnet eine TCP-Verbindung zum Player und schickt eine Nachricht
- der Player kann Nachrichten nur als Antwort auf eine solche Anfrage senden

Protokoll

$\Longrightarrow$ START(<MATCH ID>, <ROLE>, <GAME DESCRIPTION>,
<STARTCLOCK>, <PLAYCLOCK>)
(Bedenkzeit: <STARTCLOCK> s)
$\Longleftarrow$ READY

Protokoll

$\Longrightarrow$ START(<MATCH ID>, <ROLE>, <GAME DESCRIPTION>,
<STARTCLOCK>, <PLAYCLOCK>)
(Bedenkzeit: <STARTCLOCK> s)
$\Longleftarrow$ READY

$\Longrightarrow$ PLAY(<MATCH ID>, <PRIOR MOVES>)
(Bedenkzeit: <PLAYCLOCK> s)
$\Longleftarrow$ <NEXT MOVE>
...

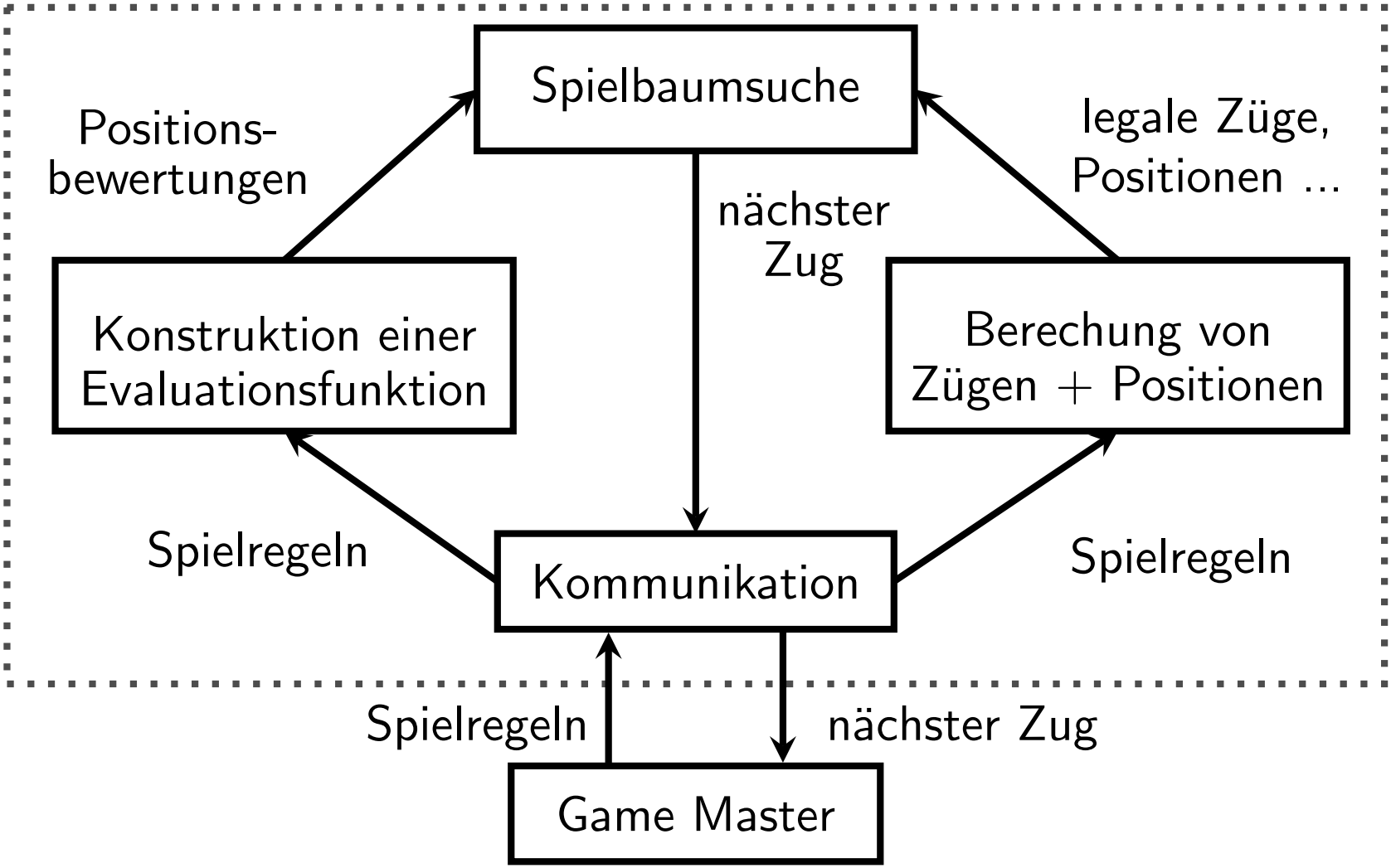
Protokoll

$\Longrightarrow$ START(<MATCH ID>, <ROLE>, <GAME DESCRIPTION>,
<STARTCLOCK>, <PLAYCLOCK>)
(Bedenkzeit: <STARTCLOCK> s)
$\Longleftarrow$ READY

$\Longrightarrow$ PLAY(<MATCH ID>, <PRIOR MOVES>)
(Bedenkzeit: <PLAYCLOCK> s)
$\Longleftarrow$ <NEXT MOVE>
...

$\Longrightarrow$ STOP(<MATCH ID>, <PRIOR MOVES>)
$\Longleftarrow$ DONE

Architektur eines General Game Players



Spielbaumsuche

1-Spieler-Spiele: SMA*, bidirektionale Suche, andere Planning-Verfahren

Spielbaumsuche

1-Spieler-Spiele: SMA*, bidirektionale Suche, andere Planning-Verfahren

2-Spieler-Nullsummen-Spiele mit abwechselnden Zügen: iterierte Tiefensuche mit Alpha-Beta

Spielbaumsuche

1-Spieler-Spiele: SMA*, bidirektionale Suche, andere Planning-Verfahren

2-Spieler-Nullsummen-Spiele mit abwechselnden Zügen: iterierte Tiefensuche mit Alpha-Beta

2-Spieler-Nullsummen-Spiele mit simultanen Zügen: von-Neumann-Minimax

Spielbaumsuche

1-Spieler-Spiele: SMA*, bidirektionale Suche, andere Planning-Verfahren

2-Spieler-Nullsummen-Spiele mit abwechselnden Zügen: iterierte Tiefensuche mit Alpha-Beta

2-Spieler-Nullsummen-Spiele mit simultanen Zügen: von-Neumann-Minimax

$\geq$3-Spieler-Spiele mit abwechselnden Zügen: Paranoid Search, Max^n

Spielbaumsuche

1-Spieler-Spiele: SMA*, bidirektionale Suche, andere Planning-Verfahren

2-Spieler-Nullsummen-Spiele mit abwechselnden Zügen: iterierte Tiefensuche mit Alpha-Beta

2-Spieler-Nullsummen-Spiele mit simultanen Zügen: von-Neumann-Minimax

$\geq$3-Spieler-Spiele mit abwechselnden Zügen: Paranoid Search, Max^n

sonstige: ???

Optimierungen

- Transpositions-Tabellen (= Hashtable / Graph-Search)
- Kombinatorische Spieltheorie

manuell konstruierte Evaluationsfunktionen

Beispiel: Materialwert bei Schach

$$h(s) = 9 \cdot n(Dame, w) + 5 \cdot n(Turm, w) + \cdots + 1 \cdot n(Bauer, w) \\ - 9 \cdot n(Dame, s) - \cdots$$

Automatische Konstruktion einer Evaluationsfunktion

- spielunabhängige Evaluationsfunktionen: Mobility, Novelty
- logische Transformationen + Reinforcement Learning
- Fuzzy goal distance

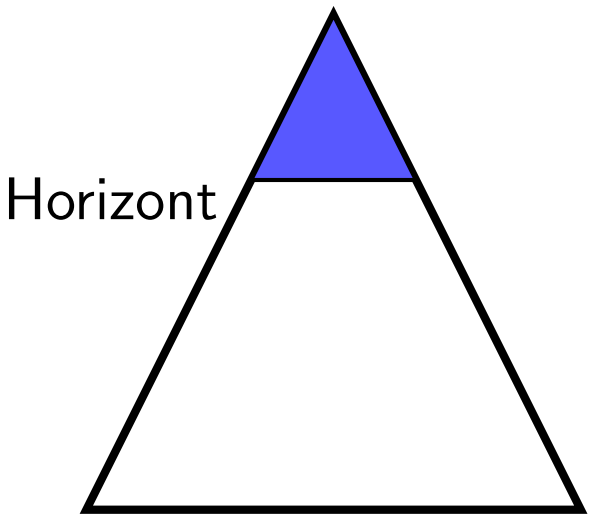
Andere Metagaming-Techniken

Metagaming

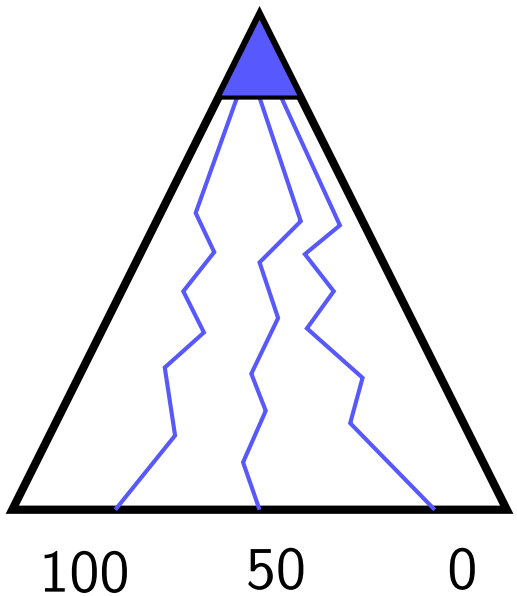
Metagaming = Schließen über Eigenschaften von Spielen

- Faktorisierung von Spielen
- Erkennung von Symmetrien
- Regeloptimierung zur schnelleren Berechnung von Positionen und Zügen: rule ordering, conjunct ordering
- Opponent Modelling (gegen "dümmere" Gegner)
- ...

Monte Carlo Tree Search (1)



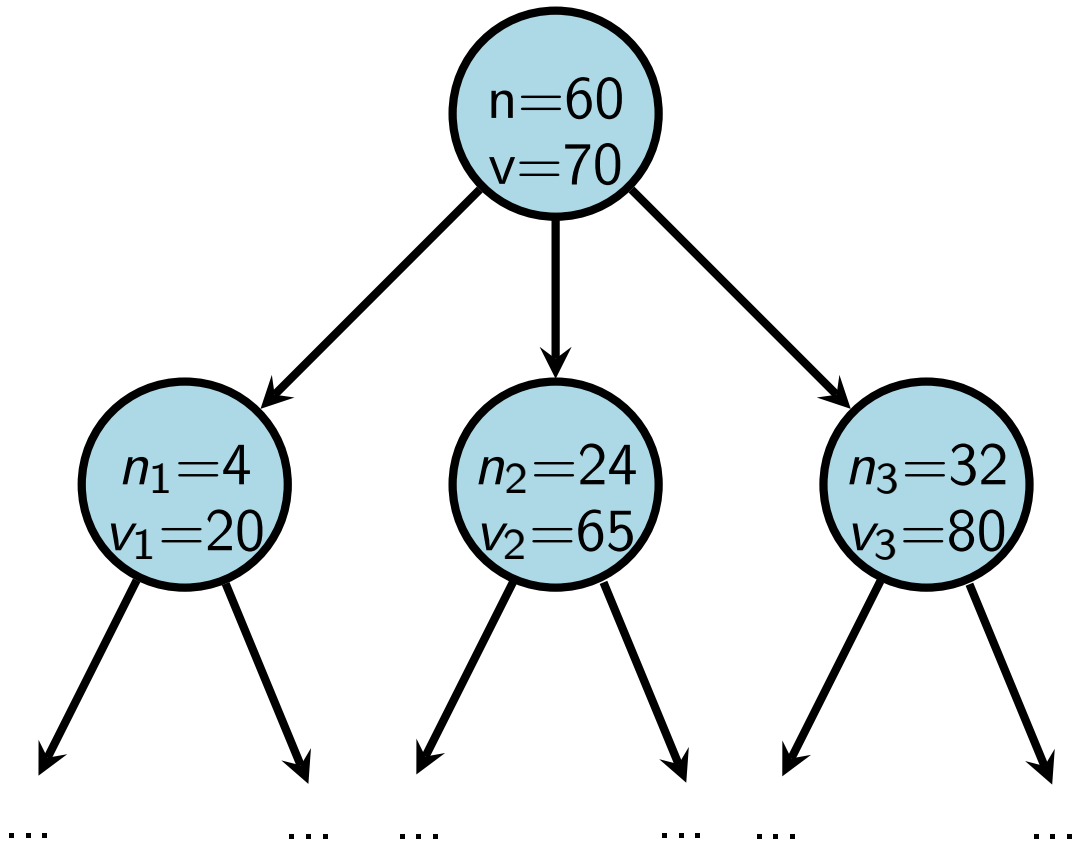
Spielbaumsuche



Monte Carlo Tree Search

Monte Carlo Tree Search (2)

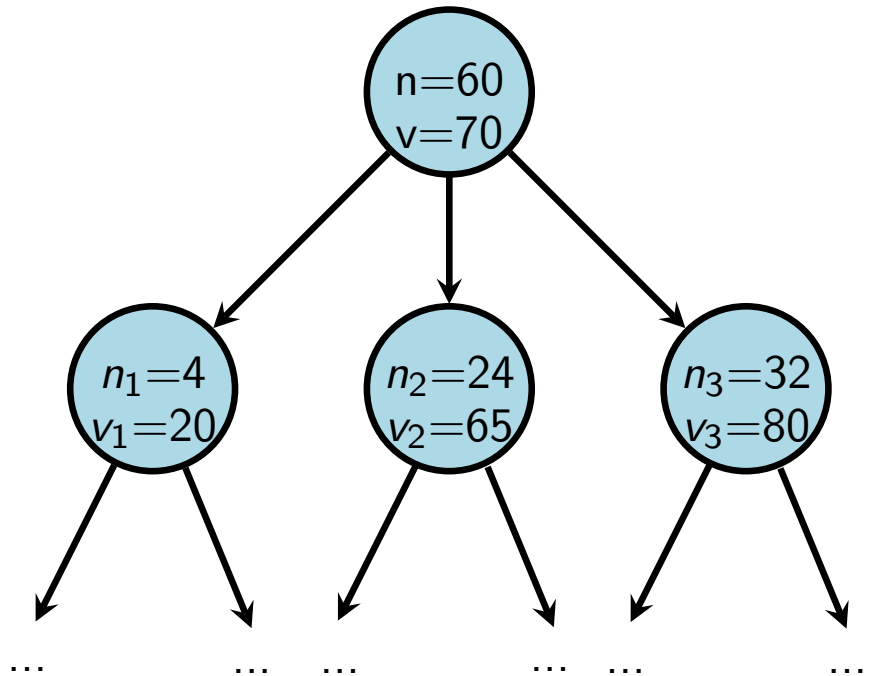
Wert eines Zuges = Mittelwert der zufälligen Playouts



Monte Carlo Tree Search (3)

- falls Zug mit $n = 0$: spiele Playout für diesen Zug
- sonst: spiele Playout für den Zug i:

$$\arg\max_i \left(v_i + C \cdot \sqrt{\frac{\log n}{n_i}} \right)$$



Weitere Infos

Weitere Informationen: `http://www.general-game-playing.de`