# Rule-Based Systems

Technische Universität Darmstadt
Knowledge Engineering Group

Seminar: Knowledge Engineering und Lernen in Spielen,
Prof. Johannes Fürnkranz, SS06

Florian Dautermann

Darmstadt, 23. Mai 2006

# Übersicht

1. Motivation
2. Informationsgewinnung (*Intelligence*)
3. Vorhersage von Spielzügen (*Prediction*)
4. Bewertung
5. Quellen

# Übersicht

1. Motivation
2. Informationsgewinnung (*Intelligence*)
3. Vorhersage von Spielzügen (*Prediction*)
4. Bewertung
5. Quellen

# Motivation

- in vielen Spielen ist es essentiell, gegnerisches Verhalten oder den Grad der gegnerischen Entwicklung zu erahnen
- menschliche Spieler ziehen aufgrund von Beobachtungen Schlussfolgerungen
- Rule Based Systems versuchen dieses menschliche Verhalten zu simulieren
- bekannt als *Expert Systems* aus nicht-Spiel-Anwendungen

# Motivation
## Beispiel: Echtzeitstrategie - Starcraft

# Motivation
## Beispiel: “Prügelspiele” - Mortal Combat

# Motivation
## Beispiel: Ego-Shooter - Quake

SOAR-Quake-Bot

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

# Übersicht

1. Motivation
2. Informationsgewinnung (*Intelligence*)
3. Vorhersage von Spielzügen (*Prediction*)
4. Bewertung
5. Quellen

# Informationsgewinnung
## Idee


Dragoon
↓ benötigt
Gateway
↓ benötigt
Nexus

# Informationsgewinnung
## Idee


Shuttle
↓ benötigt
Robotics Facility
↓ benötigt
Cybernetic Core
↓ benötigt
Gateway
↓ benötigt
Nexus

# Informationsgewinnung
## Idee


Reaver
- benötigt

Robotic Support Bay
- benötigt

Robotics Facility
- benötigt

Cybernetic Core
- benötigt

Gateway
- benötigt

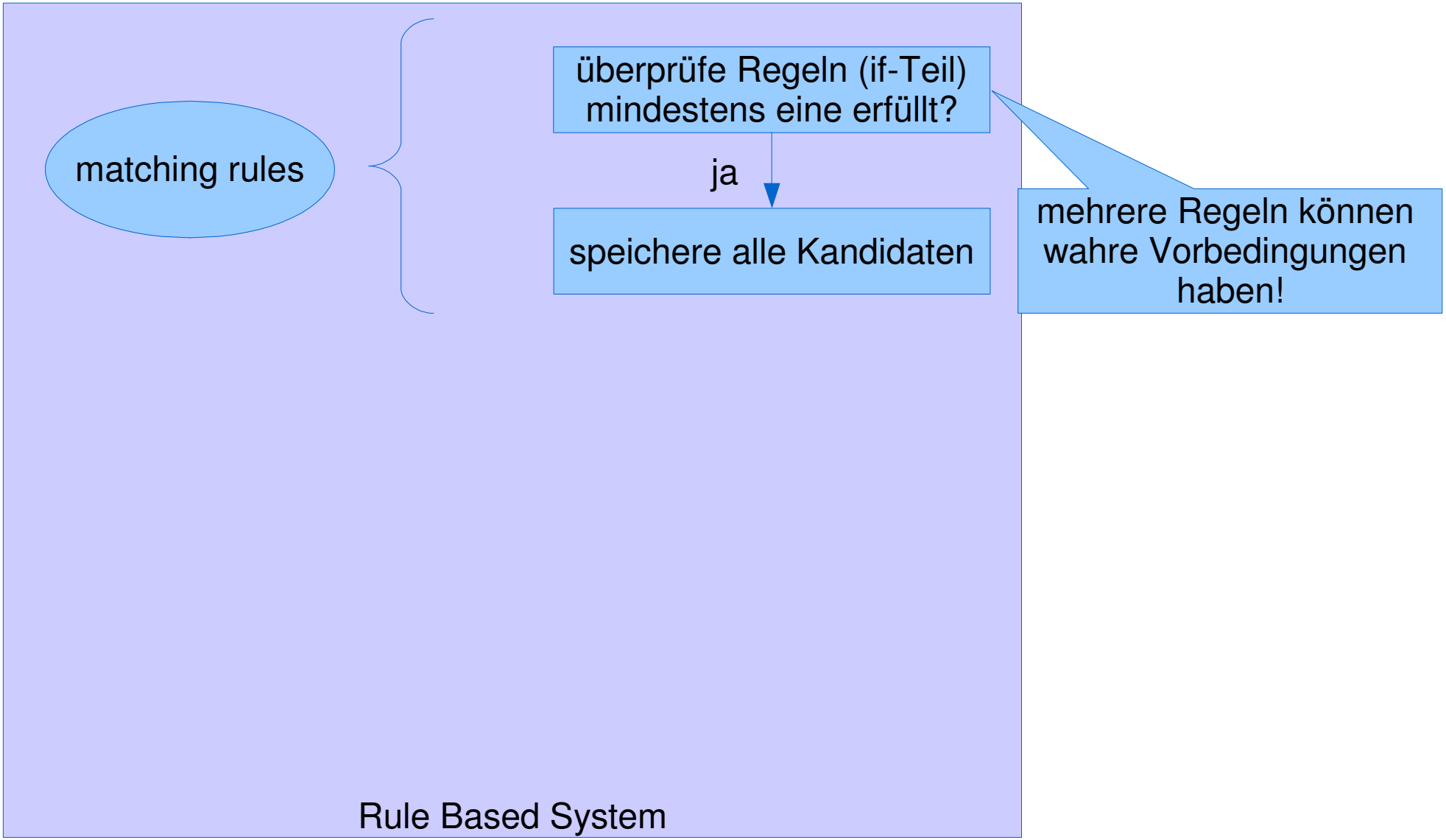
Nexus

# Informationsgewinnung
## Datenstrukturen

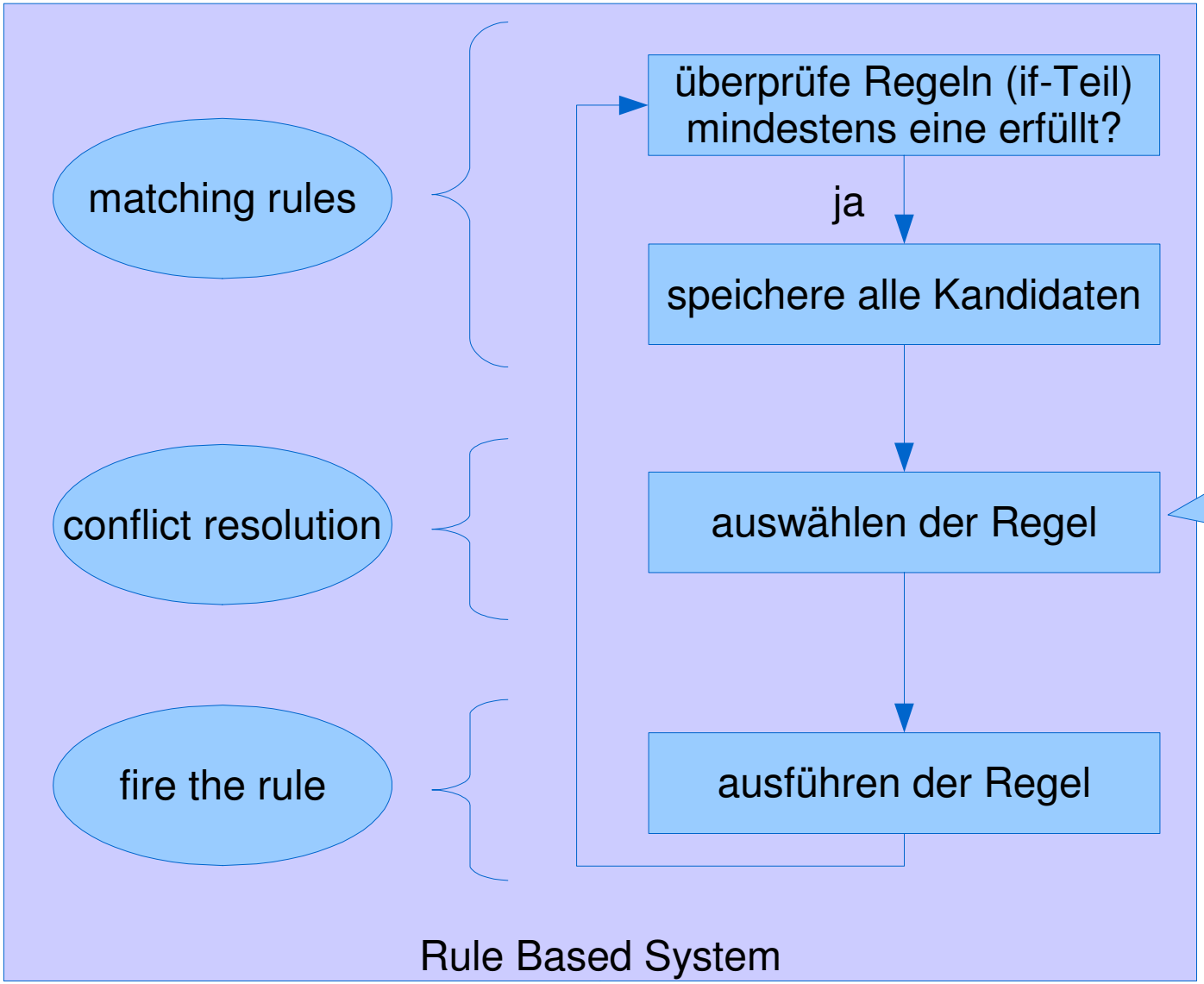
- *Working Memory*
  - Fakten & Annahmen durch Regeln
    - Bsp: `Reaver = Yes; Zealot = Maybe; Scout = Unknown`
- *Rules Memory*
  - Regeln (if-then-Stil)
    - repräsentieren den *Technology Tree*
    - Bsp:

```
if (Dragoon == Yes) {Gateway = Yes; Nexus = Yes};
if (Gateway == Yes && Zealot == Unknown) Zealot = Maybe;
```

# Informationsgewinnung
## Forward Chaining – 3 Phasen Modell

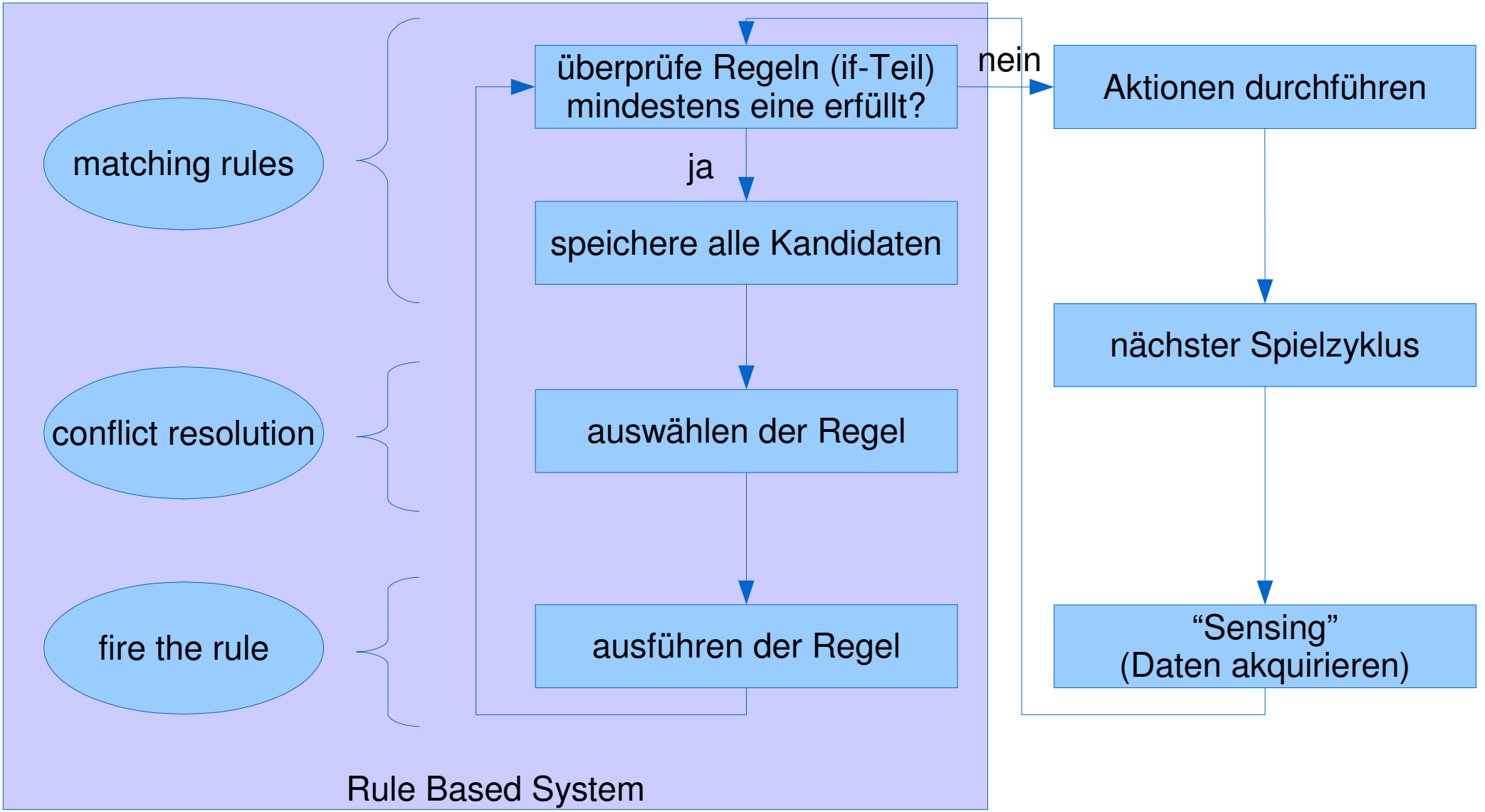


# Informationsgewinnung
## Forward Chaining – 3 Phasen Modell




Verfahren:
- First Applicable
- Random
- Most Specific
- Least Recently Used
- “Best”

# Informationsgewinnung
## Forward Chaining – 3 Phasen Modell



# Informationsgewinnung
## Backward Chaining

- Gegenteil des *Forward Chaining*
  - Benutzung des *then*-Teils um passende Regeln zu finden
    - Bsp: `if (Robotics Facility == Yes) Shuttle == Maybe`
    - Wenn überprüft werden soll, ob es ein Shuttle geben kann, muss überprüft werden ob es eine Robotics Facility gibt
  - *B.C.* ist schwerer zu implementieren als *F.C.*, vor allem mit fest programmierten if-then-Regeln
  - wird angewendet bei
    - viele Fakten, wenige Regeln
    - viele Anfangszustände, wenige Zielzustände

# Informationsgewinnung
## Laufzeitverbesserung

- Forward-Chaining hat Komplexität O((RA)^C)
  - R – Anzahl der Regeln
  - A – Anzahl der Annahmen im *Working Memory*
  - C – Anzahl der Bedingungen je Regel
- Rete-Algotrithmus zur Verbesserung
  - Merkt sich die markierten Regeln. Prüft nur gegen die Veränderungen im *Working Memory*
  - Kompexität O(RAC)
  - erhöhter Speicherbedarf

# Übersicht

1. Motivation
2. Informationsgewinnung (*Intelligence*)
3. Vorhersage von Spielzügen (*Prediction*)
4. Bewertung
5. Quellen

# Vorhersage

- Vorausahnen der gegnerischen Aktion ist essentiell in vielen Kampfspielen
- Einfaches Kampfspiel: Papier, Schere, Stein
  - kennt man die nächste Aktion kann man reagieren
  - menschliche Spieler wählen ihre Aktionen in Mustern
- RBS versuchen, diese Muster zu erkennen

# Vorhersage
## Papier, Schere, Stein - Vorführung

# Vorhersage
## Datenstrukturen

- *Working Memory*

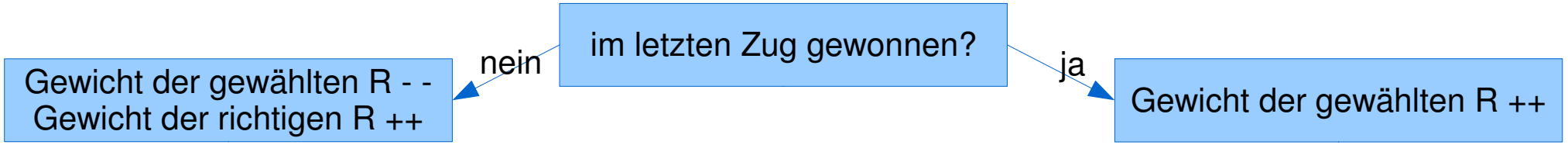
```
public class Action {
    public String act;
    public int determineWinner(Action a, Action b){...}
    public Action getCounteraction(){...} }

public class WorkingMemory {
    public Action actionA = new Action(); //vorletzte Aktion
    public Action actionB = new Action(); //letzte Aktion
    public Action actionC = new Action(); //nächste Aktion }
```

- *Rules Memory*

```
public class Rule {
    public Action antecedentA = new Action();
    public Action antecedentB = new Action();
    public Action consequentC = new Action();
    public boolean matched;
    public int weight;
    public Rule(){ matched = false; weight = 0;}
    public void setRule(Action A, Action B, Action C){...} }
```

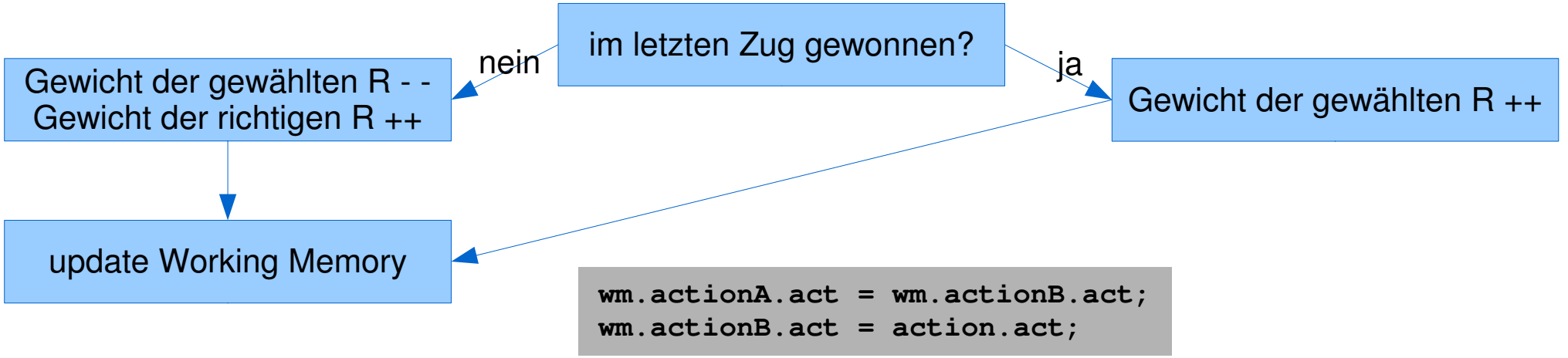
# Vorhersage
## Ablauf




```
for (i=0; i<27; i++){
  if (rules[i].matched &&
     (rules[i].consequentC.act == action.act))
           { rules[i].weight++; break; } }
```

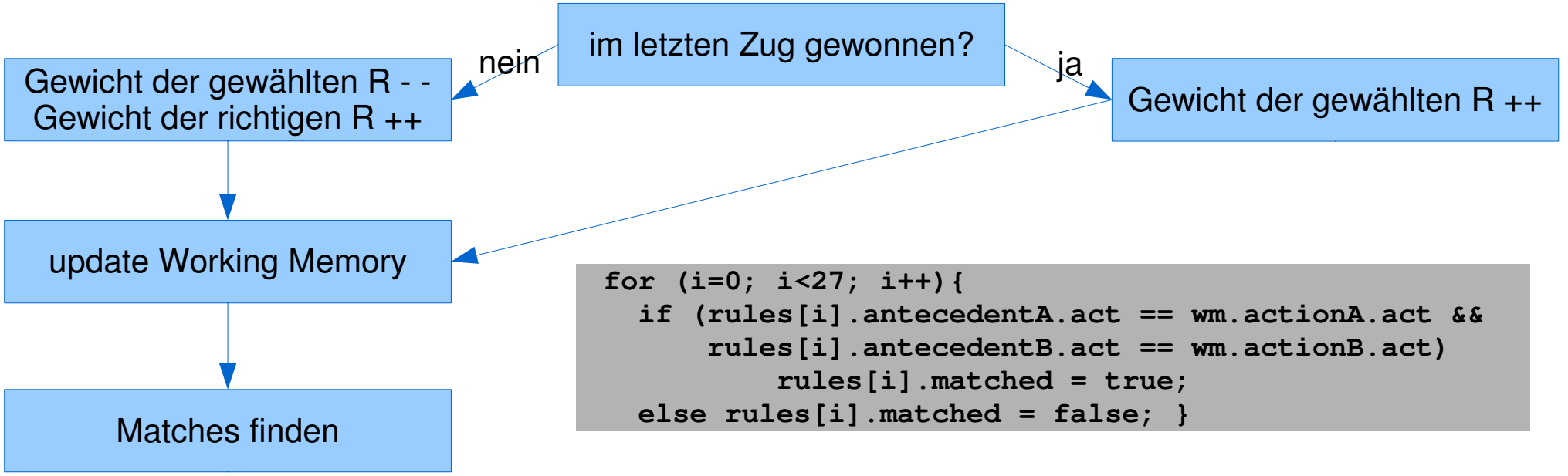
Backward Chaining!

# Vorhersage
## Ablauf



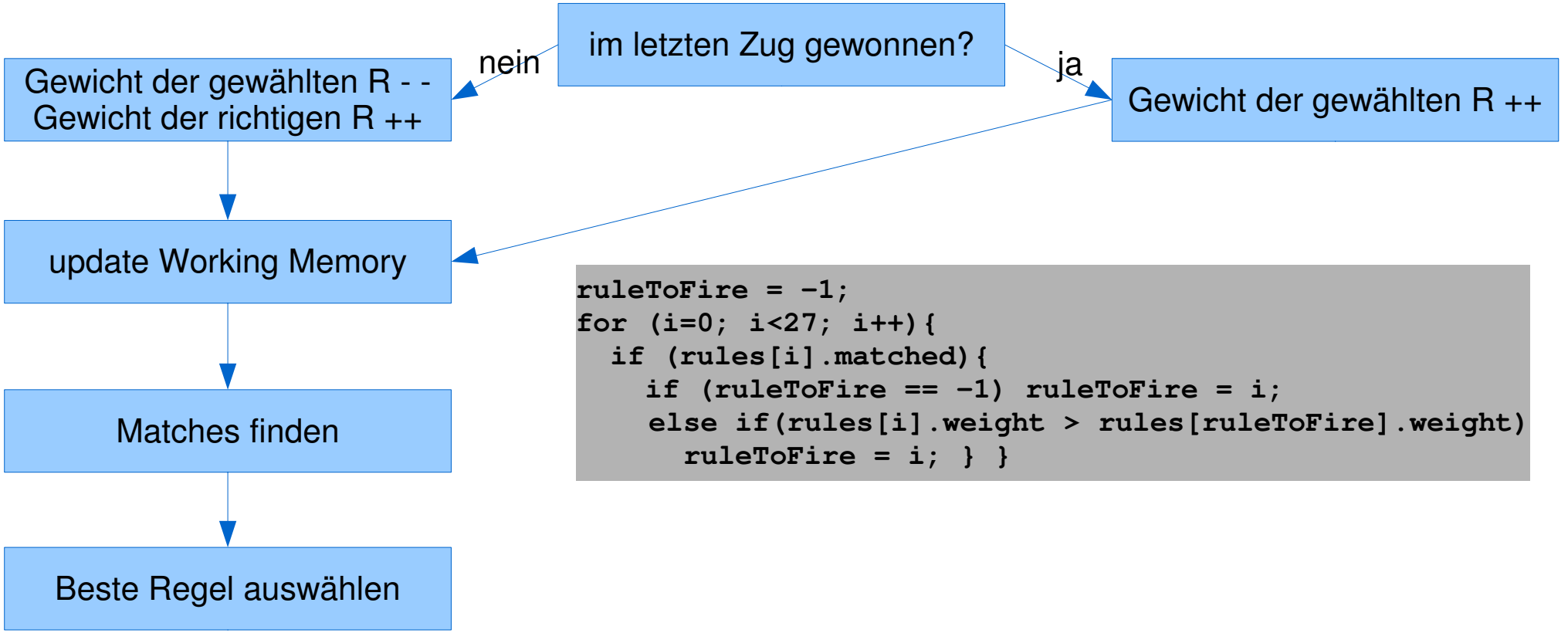

```
wm.actionA.act = wm.actionB.act;
wm.actionB.act = action.act;
```

# Vorhersage
## Ablauf



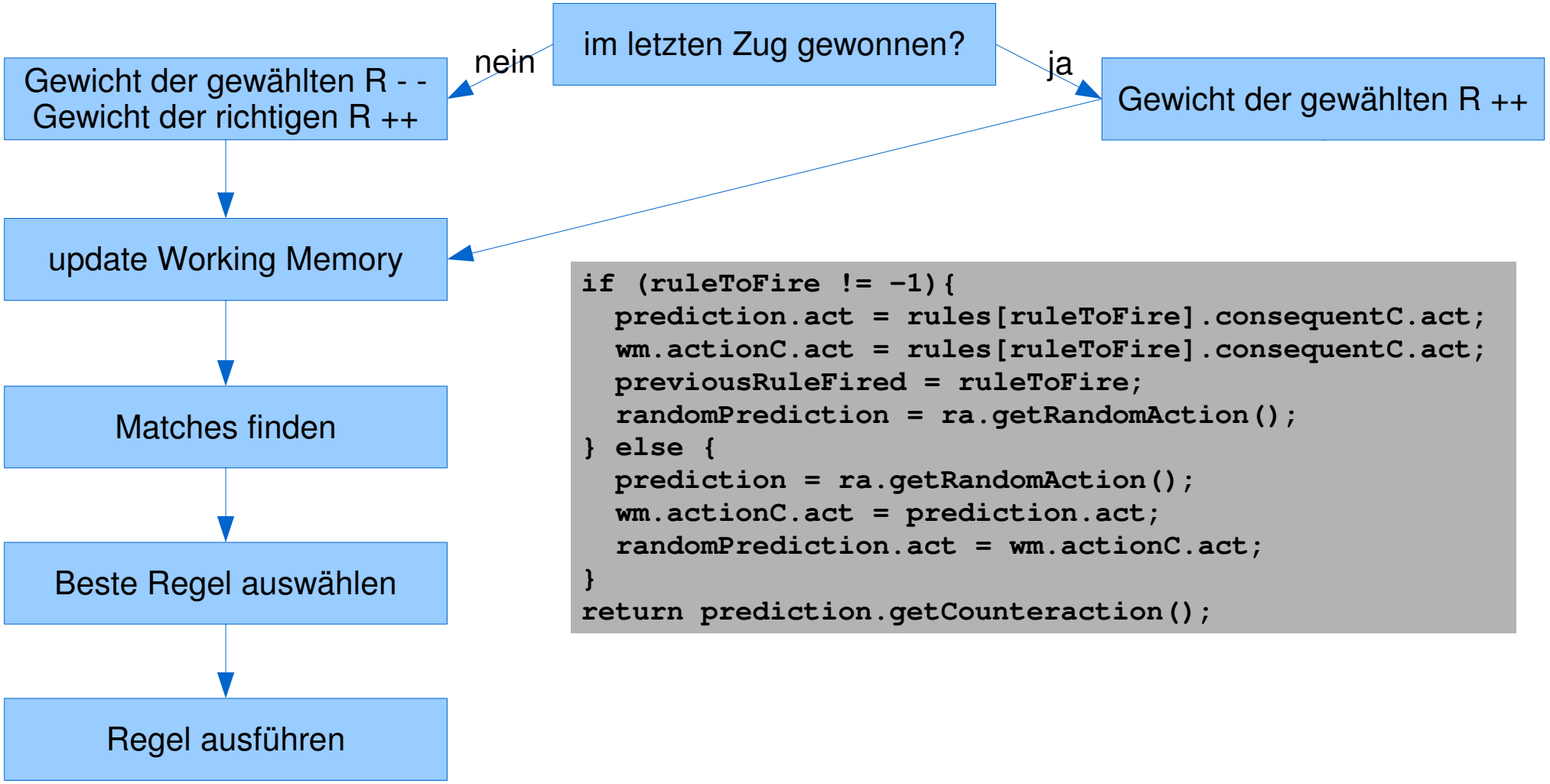

```
for (i=0; i<27; i++){
  if (rules[i].antecedentA.act == wm.actionA.act &&
      rules[i].antecedentB.act == wm.actionB.act)
          rules[i].matched = true;
  else rules[i].matched = false; }
```

# Vorhersage
## Ablauf




```
ruleToFire = -1;
for (i=0; i<27; i++){
  if (rules[i].matched){
    if (ruleToFire == -1) ruleToFire = i;
    else if(rules[i].weight > rules[ruleToFire].weight)
      ruleToFire = i; } }
```

# Vorhersage
## Ablauf




```
if (ruleToFire != -1){
  prediction.act = rules[ruleToFire].consequentC.act;
  wm.actionC.act = rules[ruleToFire].consequentC.act;
  previousRuleFired = ruleToFire;
  randomPrediction = ra.getRandomAction();
} else {
  prediction = ra.getRandomAction();
  wm.actionC.act = prediction.act;
  randomPrediction.act = wm.actionC.act;
}
return prediction.getCounteraction();
```

# Übersicht

1. Motivation
2. Informationsgewinnung (*Intelligence*)
3. Vorhersage von Spielzügen (*Prediction*)
4. Bewertung
5. Quellen

# Bewertung

- Vorteile
  - Einfachheit, basiert auf menschlichem Denken
  - Modularität, Flexibilität, einfach zu erweitern, da Regeln atomare Einheiten darstellen
  - Anwendbarkeit, funktioniert in vielen Anwendungsgebieten
  - Wissen wird von der Implementierung getrennt

# Bewertung

- Nachteile
  - geringe Aussagekraft der Regeln
  - entweder hoher Speicherbedarf oder hoher Berechnungsaufwand!
  - Anwendbarkeit, funktioniert nicht immer (z.B. wenn kein Expertenwissen vorhanden!)

# Bewertung

- Geeignete Probleme
  - Expertenwissen muss vorhanden sein
  - Auswählen der richtigen Regel sollte nicht zu wichtig sein
  - Reaktive Systeme, keine Abfolge von Aktionen

# Übersicht

1. Motivation
2. Informationsgewinnung (*Intelligence*)
3. Vorhersage von Spielzügen (*Prediction*)
4. Bewertung
5. Quellen

# Quellen


- David M. Bourg, Glenn Seemann: AI for Game Developers, O'Reilly, 2004.
- Alex J. Champandard: AI Game Development, New Riders Publishing, 2003.
- Introduction to Rule-Based Systems, James Freeman-Hargis, http://www.ai-depot.com/Tutorial/RuleBased.html
- The 2004 AIISC Report - Working Group on Rule-based Systems http://www.igda.org/ai/report-2004/rbs.html
- Johann Eder, Knowledge Engeneering http://www.isys.uni-klu.ac.at/ISYS/Courses/03WS/KE/Handouts/KE05.
- Starcraft Screenshot: http://members.lycos.nl/dracokato/starcraft.jpe
- Mortal Combat Screenshot: http://en.wikipedia.org/wiki/Mortal_Kombat_%28arcade_game%29

# Danke für Ihre Aufmerksamkeit

Fragen?