WPM Künstliche Intelligenz

Projekt: “Schiffe-Versenken”

Victor Apostel, Stefan Bogdanski und Sibille Ritter

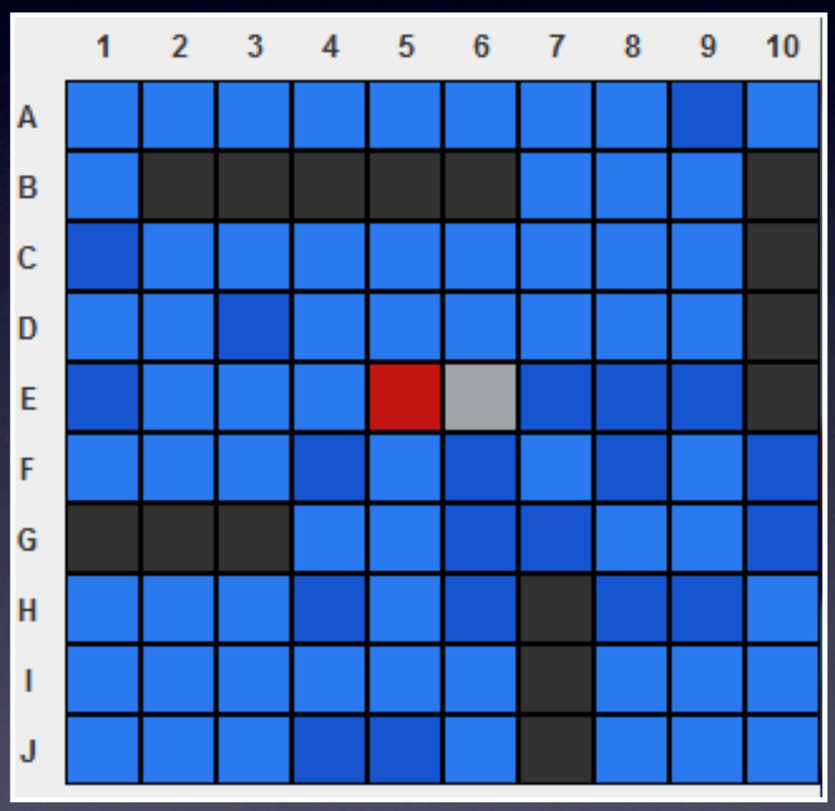
19.01.2011

Agenda

- Spielregeln (VA)
- Kommunikation (VA)
- Clientdesign (VA)
- Prolog Implementierung (SB)
- Demo: Spieler gg. KI (SR)
- Demo: KI gg. KI (SR)
- Ausblick (SR)

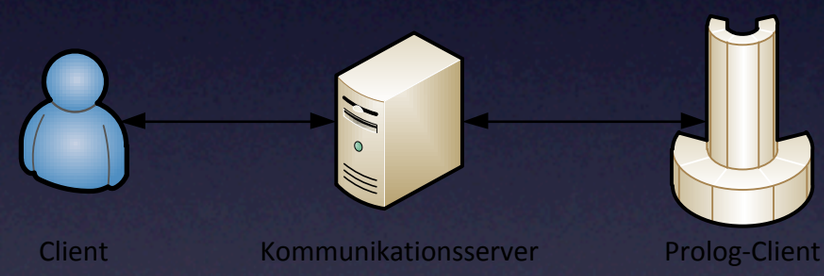
Spielregeln

- Spielfeld: 10x10
- Horizontale / Vertikale Schiffe
- 1x 5er, 1x 4er, 2x 3er, 1x 2er
- Schiffe dürfen nicht aneinander stoßen
- Abwechselndes ‘beschießen’ des Gegners



Kommunikation

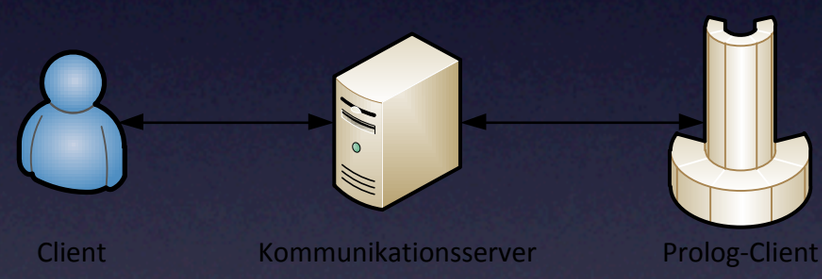
- Client-Server Architektur
- Server:
  - koordiniert Partien
  - kann beliebig viele Spiele parallel ausführen
  - dient als Nachrichtenverteiler



Serverimplementierung orientierte sich am Tic Tac Toe Beispiel und wurde in Java implementiert
Sobald sich zwei Spieler verbunden haben, wird ein eigener Thread erzeugt, der das Spiel koordiniert
Beliebige Kombinationen möglich:
    Prolog vs. Prolog
    Mensch vs. Mensch
    Prolog vs. Mensch
Koordiniert zudem, wer als erstes beginnen soll und erzeugt ein Startsignal, sobald das Spiel seitens des Servers initialisiert wurde.

Kommunikation

- Client:
  - implementiert eigentliche Spiellogik
- zwei Varianten
  - computergesteuert (Prolog)
  - HMI (Java)



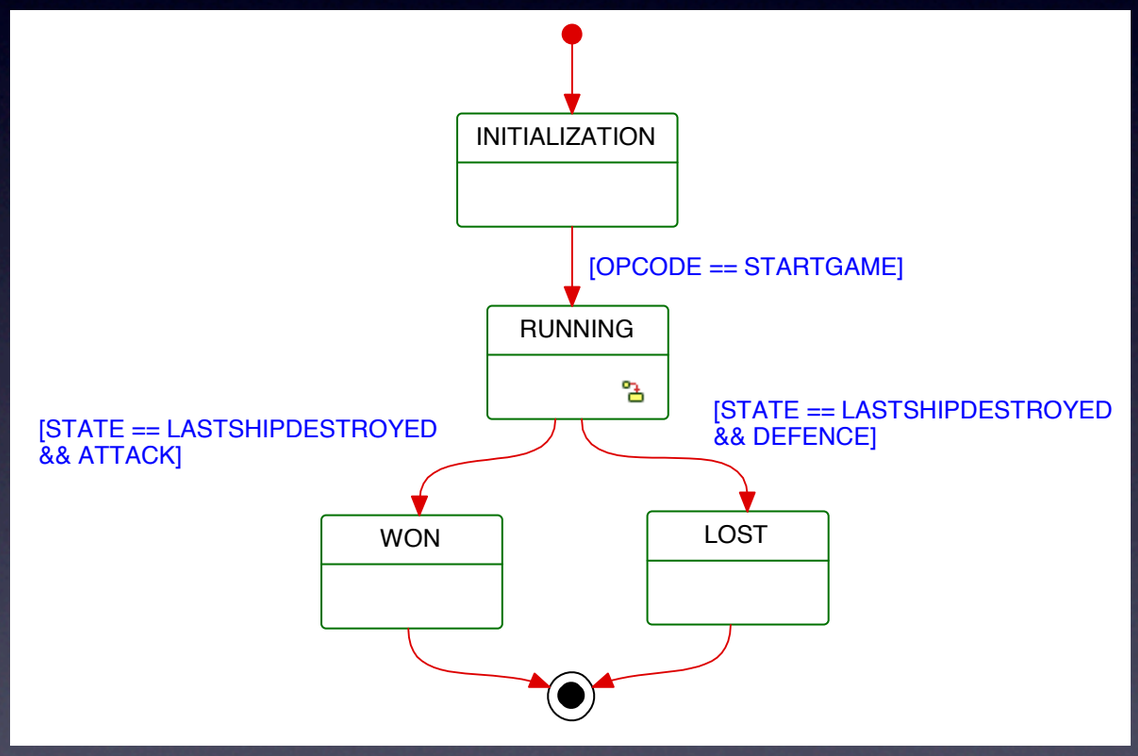
Kommunikation

- Nachrichtenaustausch auf Textbasis
- Format
  - (OPCODE, [Param 1, Param 2, Param 3]).
  - OPCODE
    - Angriff: (ATTACK, [X, Y]).
    - Antwort: (ATTACKRESPONSE, [X, Y, STATE]).
    - ...

Nachrichtenformat ist Prologorientiert
Zu erkennen an Parameterliste. Die eckigen Klammern entsprechen der Prologsyntax
Ebenso der Punkt am Ende des Kommandos
Die Parameter und Anzahl der Parameter ist vom OPCODE abhängig.
Die wichtigsten Nachrichten sind ATTACK und ATTACKRESPONSE
Die Antwort auf einen Angriff enthält einen STATE Parameter.
Dieser sagt aus, ob Wasser, oder ein Schiff getroffen wurde.
Ebenso gibts aber auch Meldungen wie „Schiff wurde zerstört“ oder „letztes Schiff wurde zerstört“

Clientdesign

- Client setzt primär Zustandsdiagramm um
- Zustandsänderungen durch Nachrichten
- Innerhalb von RUNNING Sub-Zustandsdiagramm

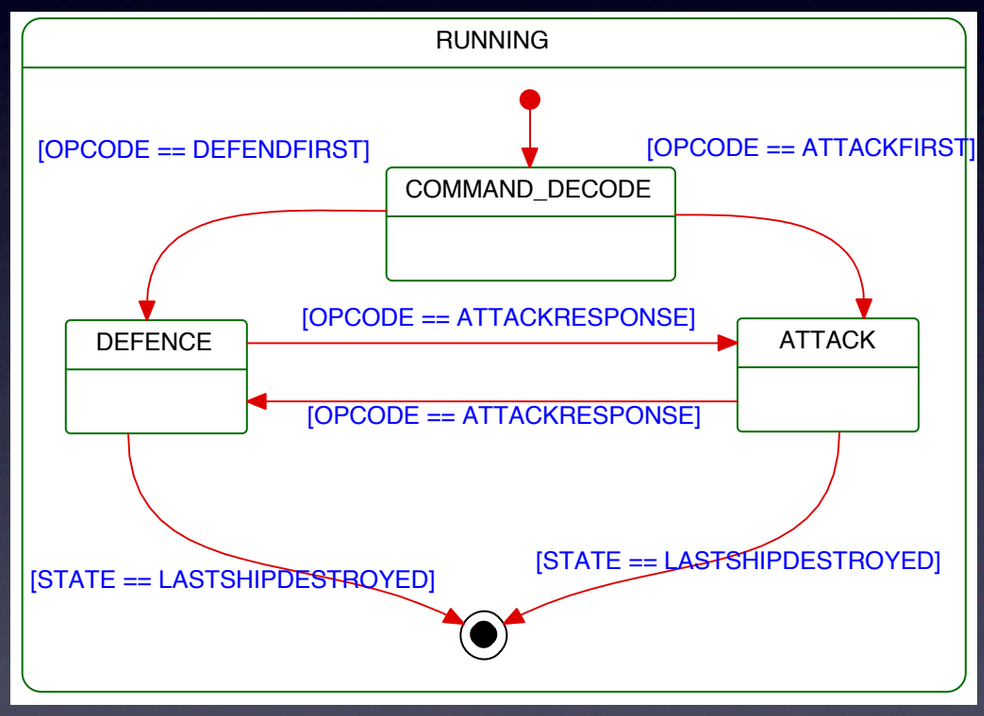


Die Kommunikation seitens des Clients kann auch als Zustandsdiagramm abgebildet werden
Zustandsdiagramm unabhängig von der Implementierung und somit allgemeingültig
Die OPCODEs und STATEs der letzten Folie dienen als Zustandsübergänge
Primär gibt es:
INITIALIZATION: Schiffe platzieren
RUNNING: Eigentliches Spielgeschehen
WON/LOST: Spielende

Innerhalb von RUNNING noch Subzustände

Clientdesign

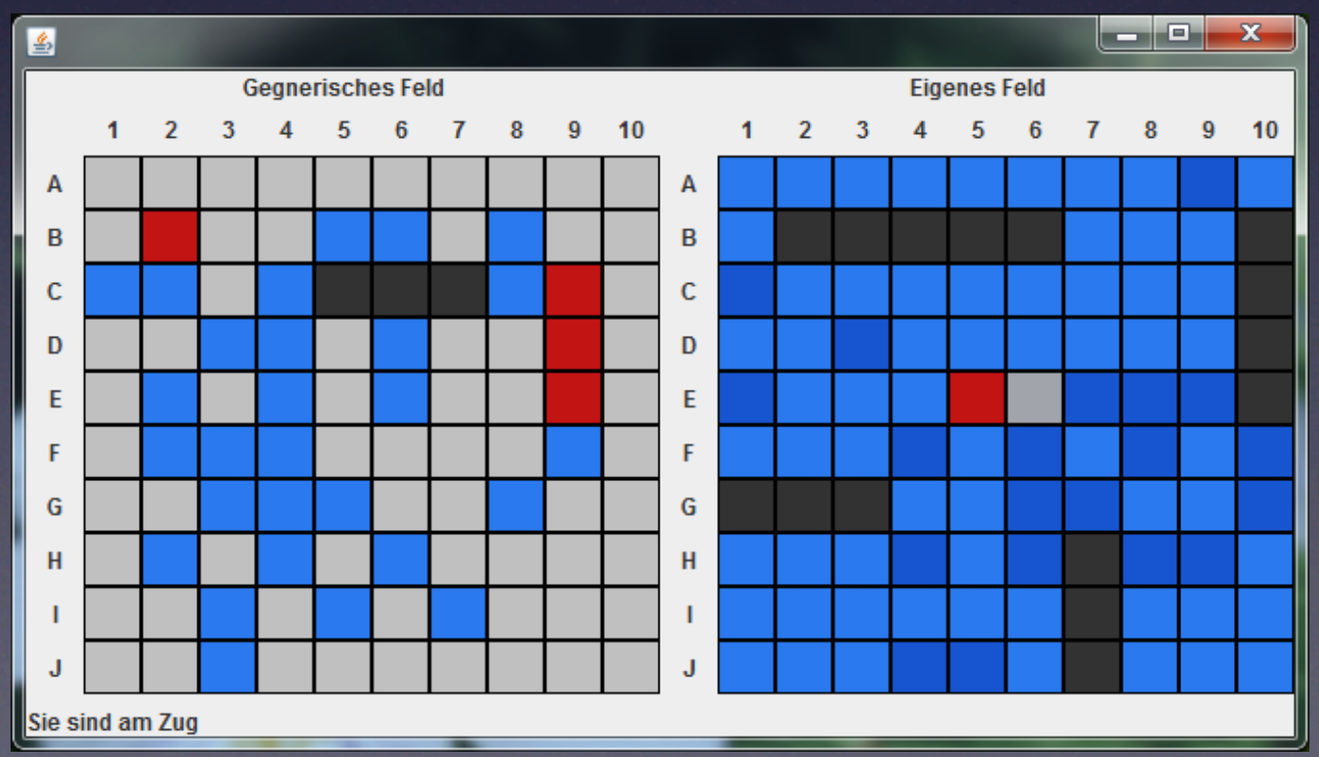
- Primäre Zustände
  - ATTACK
  - DEFENCE
- Reihenfolge der Nachrichten
- Alternierend und zum Gegenspieler entgegengesetzt



Primär ATTACK und DEFENCE
Unterscheiden sich maßgeblich durch die erwartete Reihenfolge des Nachrichtenaustauschs
    ATTACK: Angriff senden - Antwort empfangen
    DEFENCE: Angriff empfangen - Antwort senden
Nach jeder gesendeten Antwort wechselt der interne Status
Falls der STATE „letztes Schiff zerstört“ versendet wurde, beendet sich der RUNNING Zustand

Clientdesign

- Java-Client benutzt Entwurfsmuster
  - Model-View-Controller
  - Oberserver-Pattern



Die Zustandsdiagramme wurden unter Anderem in Form eines Java Clients umgesetzt
Verwendet zwei Entwurfsmuster

Die View besteht aus:
Spielfelder durch 2x 10x10 große Felder abgebildet
Links interaktiv, Rechts nur passiv
Zustand des Spielfelds durch Farben symbolisiert.
Rot: Getroffen, Blau: Wasser, dunkles Grau: Versenkt, helles Grau: noch unbekannt
immer wenn der hinterliegende Controller eine Änderung des Spielfelds vornimmt, wird die GUI automatisch aktualisiert

Die Prologseitige Realisierung des Clients stellt euch nun Stefan vor.

Prolog Implementierung I:
Spielregeln

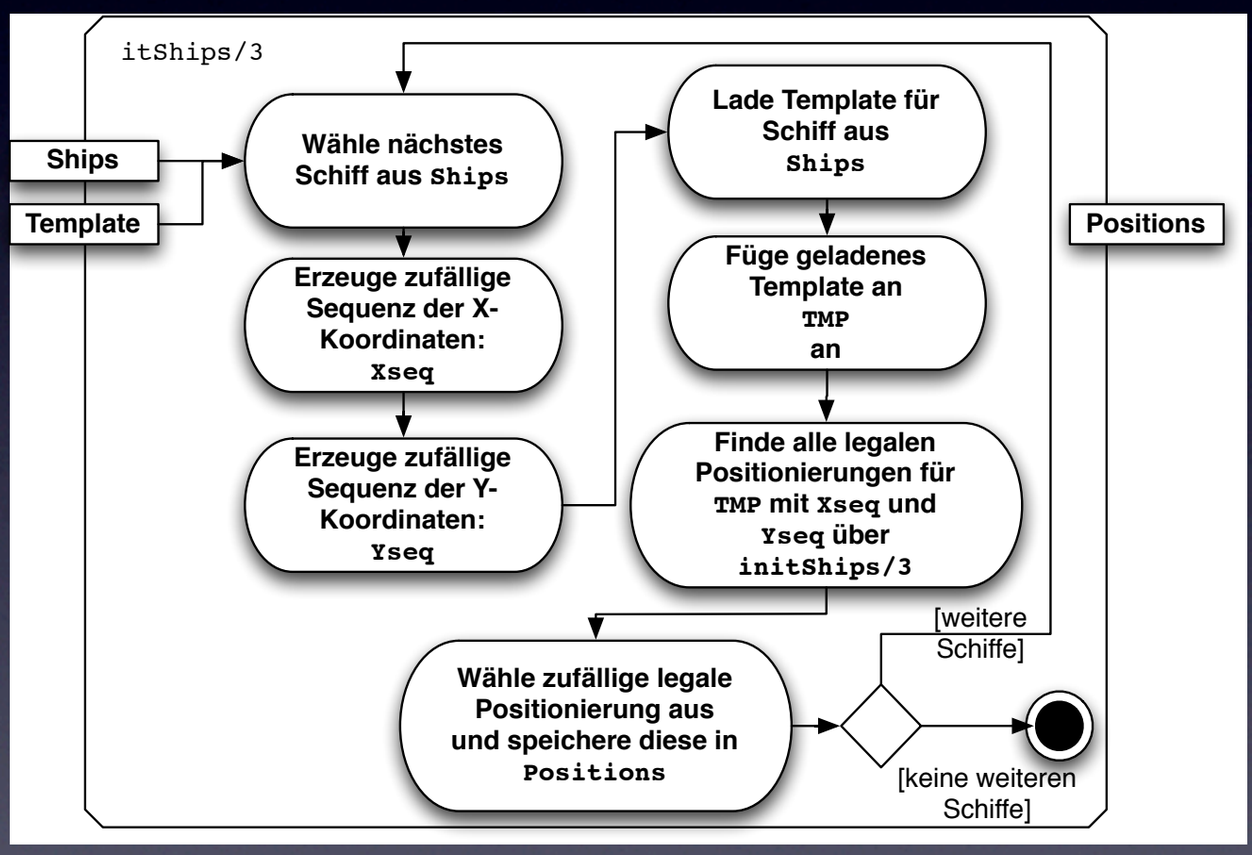
- “Schiffe müssen Horizontal (oder Vertikal) stehen”
- Schiffe dürfen einander nicht berühren, aber diagonal versetzt stehen.

```
connectedHV
(S1,P1,X1,Y1,S2,P2,X2,Y2):-
    S1 == S2,
    P1 \== P2,
    X1-X2 =:= P1-P2,
    Y1 == Y2,
    !
.

distance(S1,S2,X1,Y1,X2,Y2):-
    S1 \== S2,
    ((X1 == X2,
    abs(Y1-Y2)>1);
    (X1 \== X2,
    abs(Y1-Y2)>=1)),
    !
.
```

Prolog Implementierung I:
Platzierung

- Rekursiv
- **`Template`** der Schiffe
- Liste **`Ships`** mit zufälliger Reihenfolge
- Liste **`Positions`** mit konkreter Positionierung



Demo 1:
Spieler gg. KI

- Sibille klickt und Spricht?!

Demo II:
KI gg. KI

- Sibille startet und erklärt die Konsolenausgabe?!

Ausblick

- I grew 6inches in 2 weeks :-)