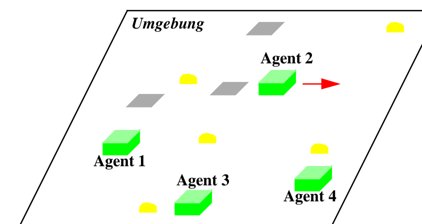
Seminar „Agentensysteme: Eine praktische Einführung“, WS 2000/2001

Einführung in die Blockwelt

3. Termin, 3.11.2000

WorldServer

- Definiert und verwaltet die Welt mit allen Agenten und Objekten (Zustand, Aktionen, etc.)
- Welt:
  - 2-dim. quadratisches Gitter mit Ursprung (0,0) in der Mitte
  - Objekte und Agenten belegen immer genau einen Gitterpunkt
  - Definiert durch Einträge in der `default.config`
  - Gittergröße: „`WorldSize 30`“, d.h. in *jede* Richtung `15` Gitterpunkte
  - „`World-KB /vol/wbs/share/Lehre/ws0001/AgentenSem/share/welten/Welt.dat`“
    ➪ Datenbank `Welt.dat`: „`(Gold gold1 7 -3) (Gold gold2 3 4)` ....“



WorldServer als WBS-Agent

- Rein reaktiver Agent, d.h. keine dynamische Welt
- Bietet zwei Dienste an (`ProvideMessage`):

1. „`World-Vis-Data`“ : Weltinformationen zur Visualisierung
   ➪Verteiler „`World-Visualization`“ ➪ Viewer-Agenten
2. „`World-Data`“: Aktuelle „Welt-Wahrnehmungen“ jedes Agenten
   ➪ Agenten

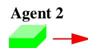
- Nimmt Aktionen der Agenten entgegen und prüft auf Durchführbarkeit
- Sämtliche Informationen als Text (`StringMessage („blablabla“)`) verschickt, der Inhalt (hier „`blablabla`“) entspricht „Blockwelt-Protokoll“

Blockwelt-Protokoll: WorldServer

Verschickt `StringMessages` Über Verteiler „`World-Visualization`“:

- Zur Initialisierung einer neuen Welt:
  „`NewWorld (<Objekt`$_1$`>) (<Objekt`$_2$`>) (<Objekt`$_3$`>)...`“, mit:
  `<Objekt`$_i$`> = „<Typ> <Name> <x> <y>`“, bei Agenten:
  `<Objekt`$_i$`> = „Agent <Agentname> <x> <y> <Richtung> <Farbe>`“, mit:
  `<Richtung>` = „`0`“ (Norden) | „`1`“ (Osten) | „`2`“ (Süden) | „`3`“ (Westen)
  `<Farbe>` = „`<R> <G> <B>`“, drei RGB-Werte von 0.0 bis 1.0, z.B.: „`0.2 1.0 0.6`“
- Aktuelle Welt: „`World (<Objekt`$_1$`>) (<Objekt`$_2$`>) (<Objekt`$_3$`>)...`“
- Wenn ein neuer Agent eingetreten ist: „`NewAgent <Agentname> <x> <y> <Richtung> <Farbe>`“
- Wenn ein Objekt (auch Agent!) gelöscht wurde: „`KillObject <Name>`“

Agenten



- Haben Position und Ausrichtung in der Welt
  ➪ Blick- und Fortbewegungsrichtung vorgegeben
- Proaktiv (`ActiveThreadAgent`) ➪ Ständige Verarbeitungsschleife

  → Wahrnehmen → Schlussfolgern → Handeln

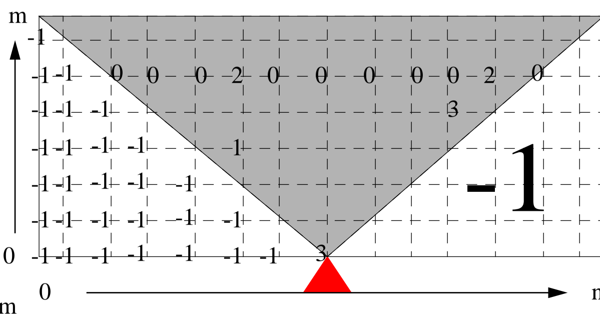
- Wartezeit zwischen zwei Zyklen in `default.config` definierbar: „`SleepDuration 1`“
- Handeln in der Welt durch Fortbewegen, genauer:
  Pro Handlungsschritt **1 x Drehen und *dann* max. einen Schritt nach vorne**!
- Drehen im eigenen („egozentrischen“) Bezugssystem, d.h. „vorwärts“, „rechts herum“, „links herum“ oder „umdrehen“

Agenten und WorldServer

- WorldServer bietet Dienst „`World-Data`“ ➪ Agenten melden sich *explizit* an:
  „`Join <Farbe>`“ (Agentenname durch Absender der Nachricht gegeben)
- WorldServer schickt Nachricht über Startposition/-ausrichtung des Agenten in der Welt:
  „`Start <x> <y> <Richtung>`“
- Agenten senden ihre Aktionen an den WorldServer
  Achtung: Erfolg *nicht* garantiert und auch *nicht* mitgeteilt!
- WorldServer schickt *aktuelle* Informationen über die Welt, d.h. bei *jeder* Änderung der Welt!

Agenten: Wahrnehmen

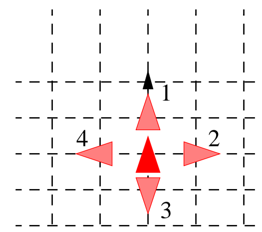
- WorldServer unterrichtet Agenten über die Welt: *Individuelle* simulierte *visuelle* Wahrnehmung:
  „`World <Weltausschnitt>`“, mit:
  `<Weltausschnitt>` = „`#Zeilen #Spalten` $x_{00} \ldots x_{0n}$ $x_{10} \ldots x_{1n}$ $\ldots$ x_{mn}“
  Die (x_{ij}) beschreiben den wahrgenommenen Sichtkegel.
- Wert x_{ij} gibt das Objekt an den Koordinaten (resp. auf dem Gitterpunkt) an:
  -1 ➪ unbekannt
  0 ➪ leer
  1 ➪ Loch
  2 ➪ Gold
  3 ➪ Agent
  ... ➪ (wird noch nicht verraten)
  99 ➪ Rand (unerlaubtes Gebiet)



- Grösse des Sichtkegels festgelegt in `default.config`:
  Eintrag: „`VisDistance 6`“ ➪ m=6, n=2*m

Agenten: Handeln

- Bietet Dienst „`Agent-Data-Service`“ (über Verteiler „`Agent-Data`“) an
  ➪ Alle Aktionen über Verteiler „`Agent-Data`“
  ➪ WorldServer (trägt sich ein)
- Handelt in der Welt durch *Fortbewegen* ➪ Nachricht an „`Agent-Data`“-Verteiler:
  „`Move <Richtung> <Schritt>`“, mit:
  `<Richtung>` = 1 (vorwärts), 2 (rechts herum), 3 (umdrehen), 4 (links herum)
  `<Schritt>` = 0 (keinen Schritt), 1 (einen Schritt)



Wie baue ich meinen eigenen Agenten?

- Beispiel-Agent unter `.../agenten/TestAgent/src/`
- Eine „Schablone" für den eigenen Agenten anlegen:
  1. Lokales Arbeitsverzeichnis (irgendwo im Homes) erstellen
  2. Die Daten „`createAgentDir`" und „`AgentDirTemplate`" aus dem Unterzeichnis `.../agenten/` kopieren
  3. Ausführen: „`createAgentDir <Agentname>`"

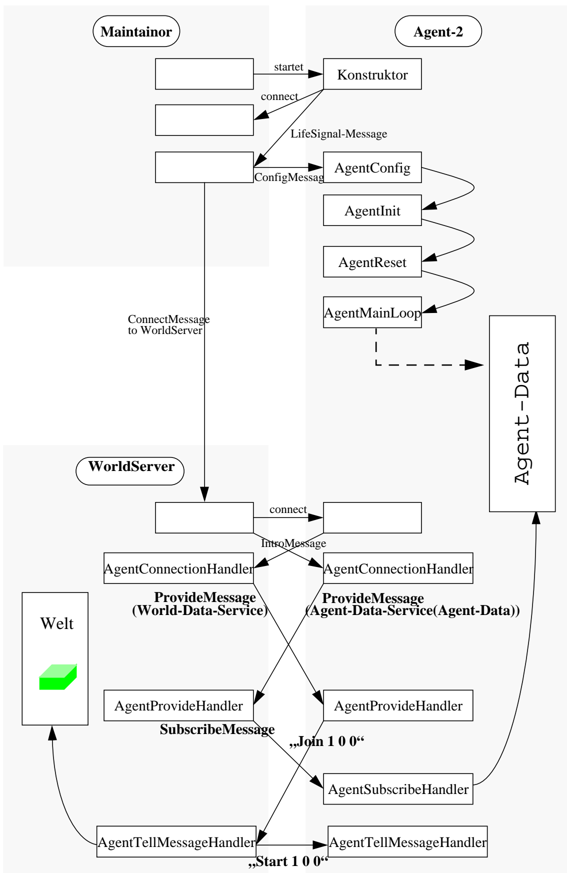
⇧ Sourcen:
1. `<Agentname>_Main.cc` (ergibt das Executable, unwichtig)
2. `<Agentname>.hh`
3. `<Agentname>.cc`

- Compilieren einfach mit „make" ⇨ Ausführbares Programm <Agentname>
- In eigenes Agentensystem einbinden über `agents.config`:

```
OWNPATH <Arbeitszerzeichnis>
....
AgentXYZ <Agentname> - - - .
```

TestAgent.cc

```
#include "TestAgent.hh"

// Konstruktor
TestAgent:: TestAgent (const char *agentname, const char *maintainhost,
                       int maintainport)
  : ActiveThreadAgent (agentname, maintainhost, maintainport)
{
  r=1.0; g=0.0; b=0.0;
  sleepDuration = 1;
 blickweite = 1;
  interface->createDistributor("Agent-Data");
}

// Destruktor
TestAgent::~TestAgent()
{}

// Initialisierung des Agenten
void TestAgent::AgentInit()
{}

// Konfiguration des Agenten
void TestAgent::AgentConfig(const BO_String &slot,
                             const BO_String &value)
{
  if (slot == "VisDistance")
    blickweite = (int)value;
  else if (slot == "SleepDuration")
    sleepDuration = (int)value;
}

// Zuruecksetzen des Agenten
void TestAgent::AgentReset()
{}

// Wird aufgerufen, wenn neuer Kommunikationspartner hinzugekommen ist
void TestAgent::AgentConnectionHandler(const BO_String &agent)
{
  ProvideMessage("Agent-Data-Service (Agent-Data)").tell(agent);
}

// Wird aufgerufen, wenn sich ein neuer Agent auf einem
// unserer Verteiler eintraegt
void TestAgent::AgentSubscribeHandler(const BO_String &agent,
                                       const BO_String &verteiler)
{}
```



TestAgent.hh

```
#ifndef TestAgent_H
#define TestAgent_H

#include <math.h>
#include "ActiveThreadAgent.hh"

class TestAgent : public ActiveThreadAgent
{
  public :

  // Konstruktor
  TestAgent(const char *agentname, const char *maintainhost,
            int maintainport);

  // Destruktor
  virtual ~TestAgent();

  // Main-Loop
  virtual void AgentMainLoop();

  protected :

  // Die benoetigten Methoden fuer den Agenten
  virtual void AgentInit();
  virtual void AgentReset();
  virtual void AgentConfig(const BO_String &slot, const BO_String &value);
  virtual void AgentConnectionHandler(const BO_String &agentname);
  virtual void AgentProvideHandler(const BO_String&, const BO_String&,
                                   const BO_String&);
  virtual void AgentSubscribeHandler(const BO_String&, const BO_String&);
  virtual void AgentTellMessageHandler(Message *msg);

  // Soll ein neuer Move gemacht werden? Wenn ja, welcher?
  bool howToMove(int&, int&);

  // Einen Move vollziehen
  void doMove(int, int);

  // Daten
  int welt [25][25];
  int blickweite;
  float r,g,b;
  int sleepDuration;
};
```

```
// MainLoop des Agenten
void TestAgent::AgentMainLoop()
{
  int direction, step;

  // Hauptschleife des Agenten: perceive -> reason -> act
  // (perceive geschieht nebenlaeufig durch Nachrichtenempfang)
  while (true) {
    sleep(sleepDuration);

    // reason: wie sieht der naechste move aus?
    if (howToMove(direction, step))

      // act: vollfuehre move
      doMove(direction, step);
  }
}

// Reason-Methode des Agenten: Waehle naechste Aktion
bool TestAgent::howToMove(int &direction, int &step)
{
  // zufaelliges Umherlaufen
  int i = rand();
  i = i/8191.75 + 1;
  direction = i;
  step = 1;
  switch (direction) {
  case 1:
    cout << "Weiter vorwaerts" << endl;
    break;
  case 2:
    cout << "Drehe nach rechts" << endl;
    break;
  case 3:
    cout << "Drehe um" << endl;
    break;
  case 4:
    cout << "Drehe nach links" << endl;
    break;
  }
  return true;
}
```



```
// Ein anderer Agent bietet einen bestimmten Dienst an.
// Entsprechende Nachrichten gehen bei diesem ueber einen
// zugehoerigen Verteiler.
void TestAgent::AgentProvideHandler(const BO_String &agent,
                                    const BO_String &dienst,
                                    const BO_String &verteiler)
{
  // Beim WorldServer anmelden/eintragen
  if (dienst == "World-Data-Service") {
    BO_String s;
    s << "Join " << r << " " << g << " " << b << STRING_END;
    StringMessage(s).tell(agent);
  }
}

// Verarbeitung eingehender Nachrichten
void TestAgent::AgentTellMessageHandler(Message *msg)
{
  if (msg) {
    if (msg->getID() == "STR") {
      StringMessage *smsg = dynamic_cast<StringMessage*>(msg);
      if (smsg) {
        BO_String ident, content;
        content.instream_line();
        smsg->string >> ident >> content;
        if (ident == "Start") {
          // WorldServer hat Startposition mitgeteilt
          int x,y,d;
          content >> x >> y >> d;
          cout << "Starte bei " << x << "," << y
               << ", Richtung " << d << endl;
        }
        else if (ident == "World") {
         // WorldServer hat einen neuen Weltausschnitt geliefert
          int entf,breite;
          content >> entf >> breite;
          cout << "Neues Sichtfeld ------" << endl;
          for (int i=0; i<entf; i++) {
            for (int j=0; j<breite; j++) {
              content >> welt[i][j];
              cout << welt[i][j];
            }
            cout << endl;
          }
          cout << "----------------------" << endl;
        }
        else
          cerr << "unbekannte Nachricht:" << ident << endl;
      }
}}}
```



```
// Act-Methode: Mache den Schritt
void TestAgent::doMove(int direction, int step)
{
  BO_String s;
  s << "Move " << direction << " " << step << STRING_END;
  StringMessage m (s);
  m.tellDistr("Agent-Data");
}
```

Aufgaben

- Eigenen Agenten einrichten, kompilieren und in eigenes Agentensystem aufnehmen
- Eigenen Blockwelt-Agenten programmieren
- *Für Interessierte*: Textbasierten Viewer

Nächste Woche

- Blockwelt bevölkern
- Theorie: Kommunikation mit Nachrichten
- Technischer Support (ständig: Kontakt siehe Webseite `http://www.TechFak.Uni-Bielefeld.DE/tech-fak/ags/wbski/lehre/digiSA/Agentensysteme/`)