Multi **A**gent **A**uthoring **T**ool for Programming Autonomous Mobile Robots

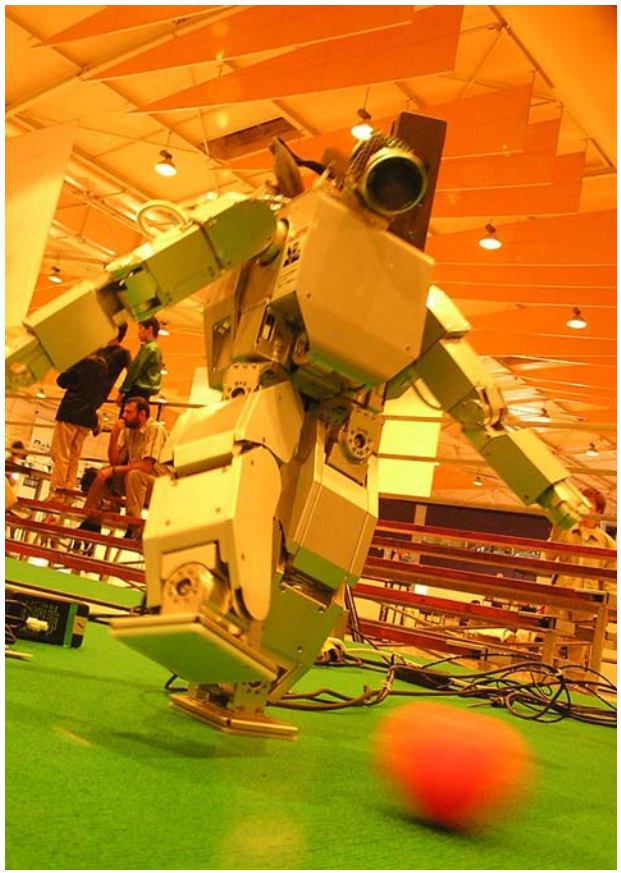
Diplomarbeit von
Anna Egorova

AG Künstliche Intelligenz
Prof. Dr. Raúl Rojas

Übersicht

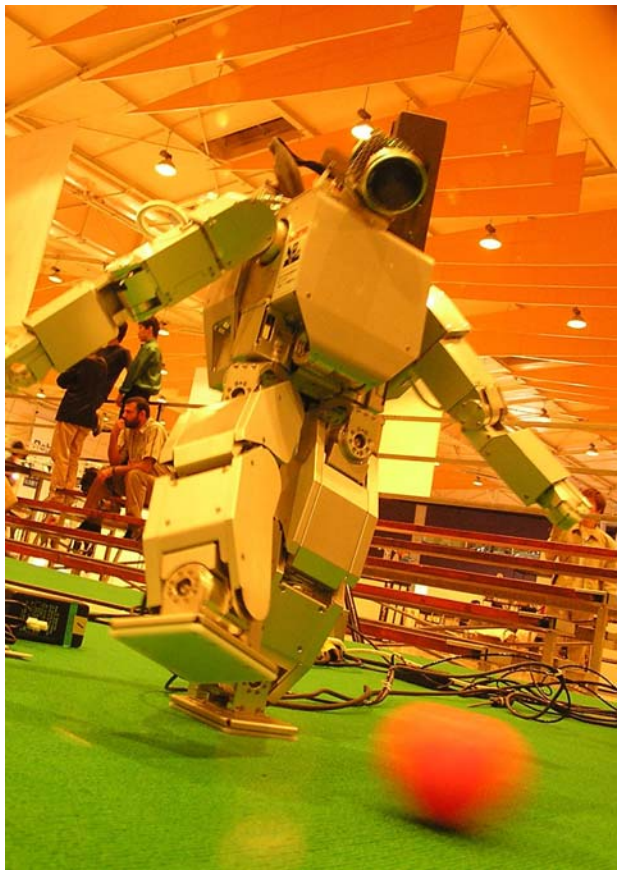
- RoboCup und FU Fighters
- Das Verhaltensmodell der FU Fighters
- Das Ziel von MAAT
- Übersicht von MAAT
- Demonstration

RoboCup Leagues

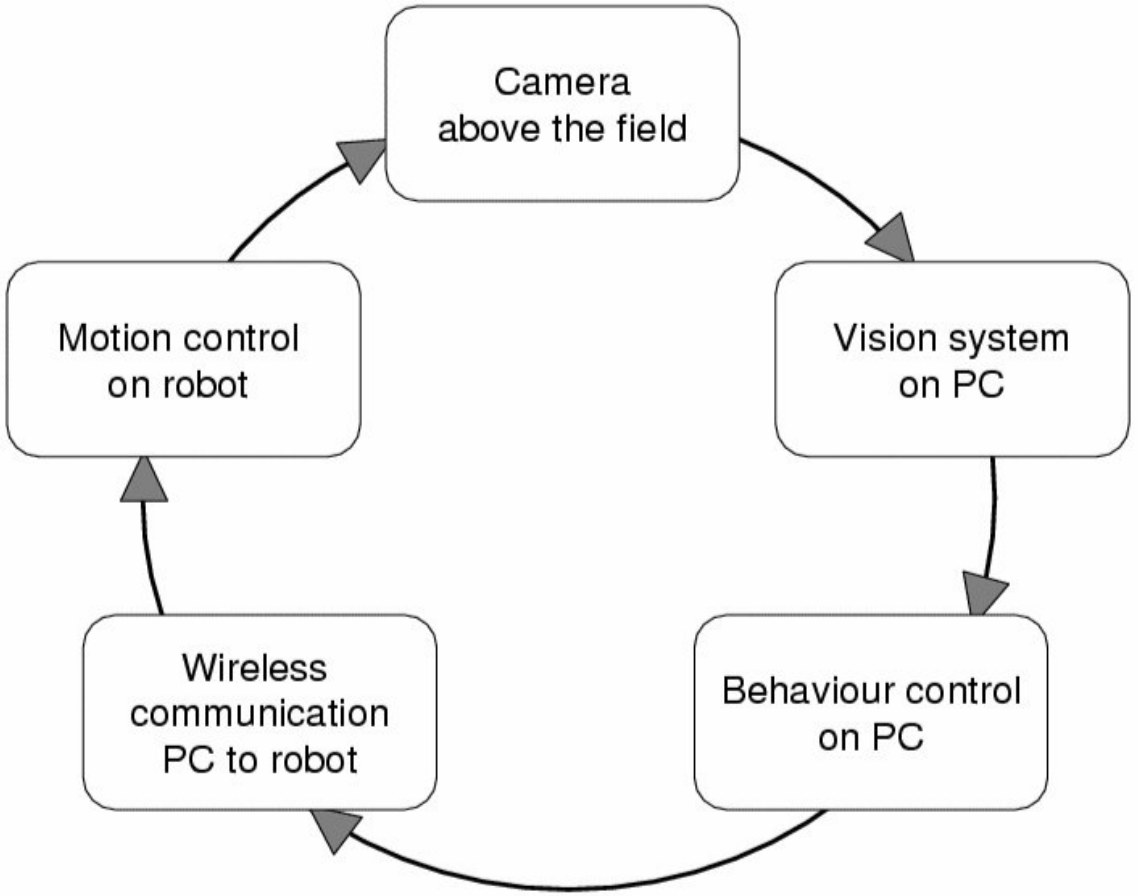


RoboCup Leagues

Die Weltmeister!



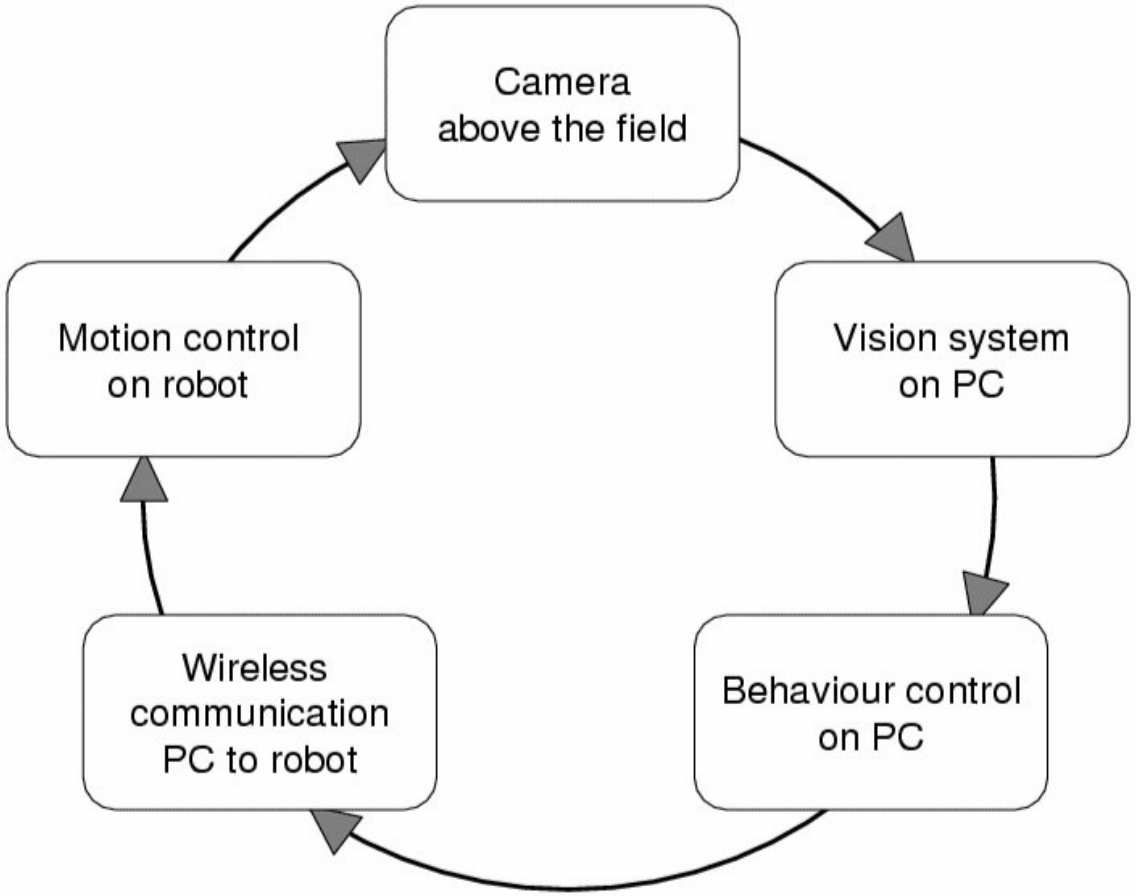
Das System der FU Fighters



Module:

- Bildverarbeitung
- Verhalten
- Kommunikation
- On-board Controller

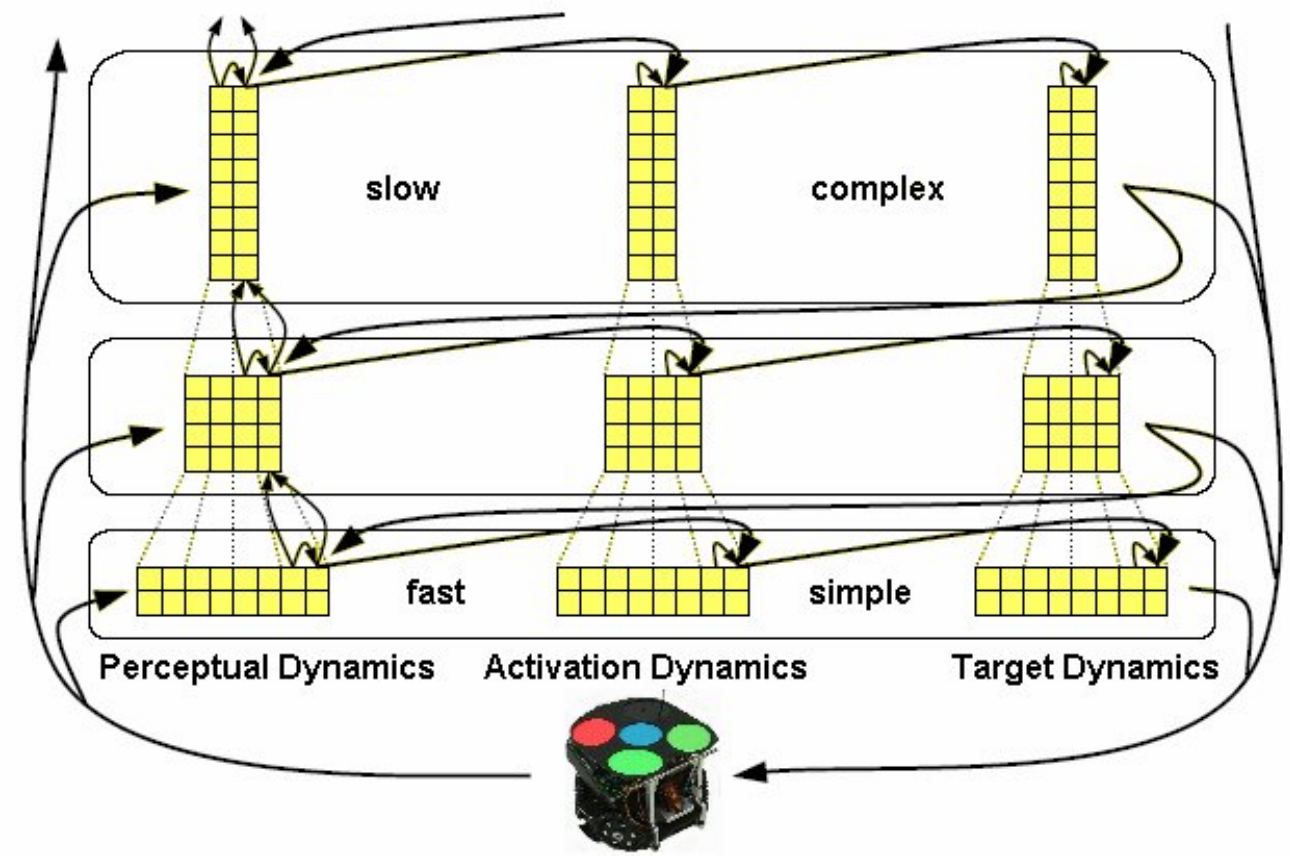
Das System der FU Fighters



Module:

- Bildverarbeitung
- **Verhalten**
- Kommunikation
- On-board Controller

Das Verhaltensmodell 1

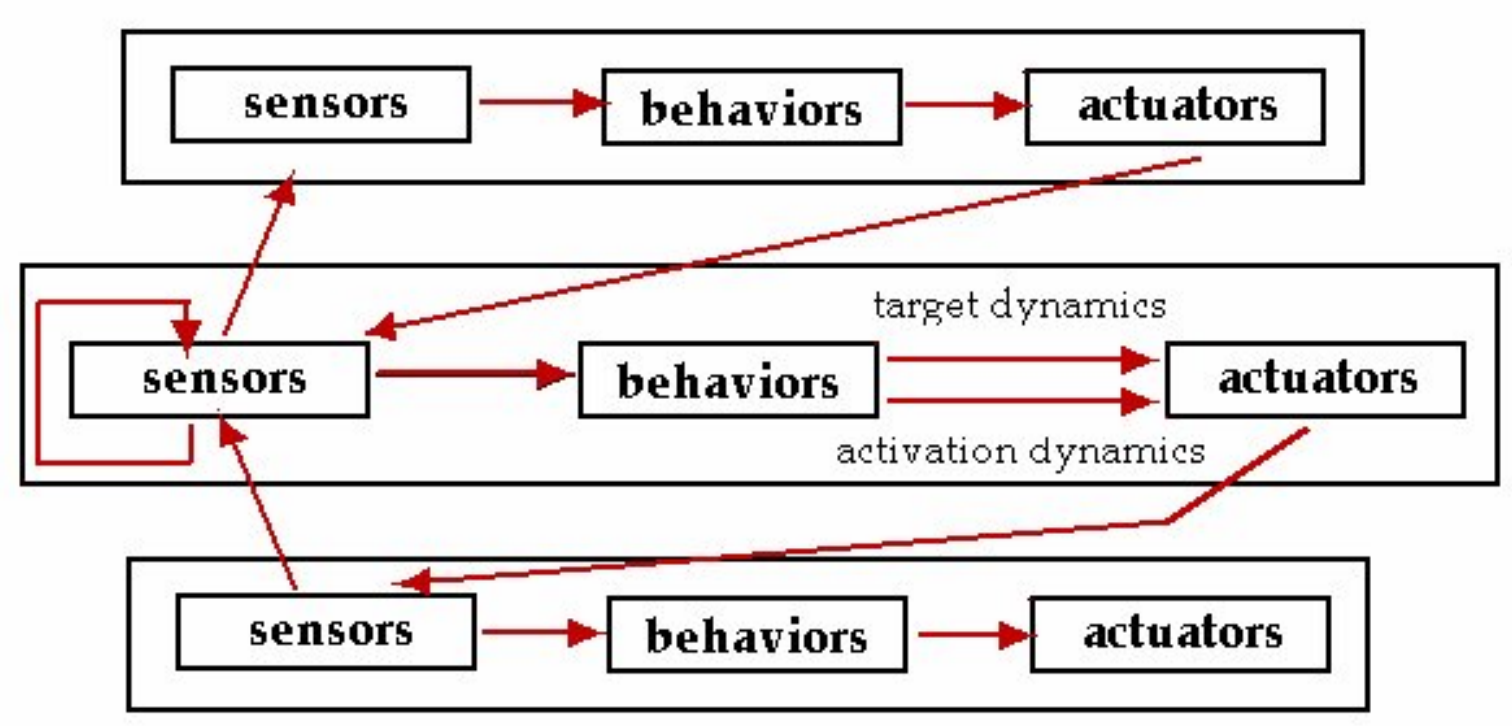


Organisiert in:

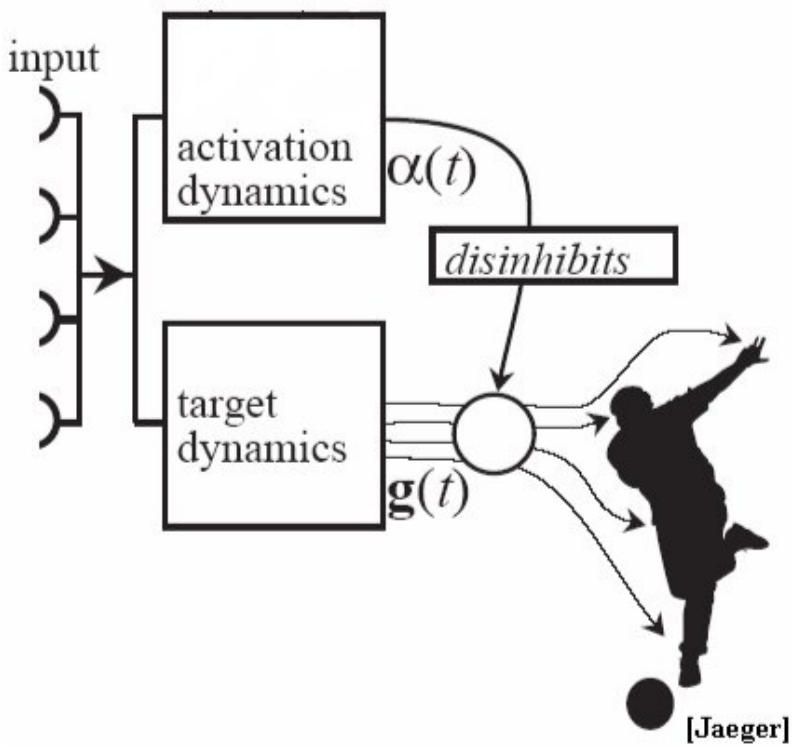
- Ebenen
- Verhalten in den Ebenen
- Sensoren
- Aktoren

Das Verhaltensmodell 2

Kommunikation zwischen Ebenen



Das Verhaltensmodell 3

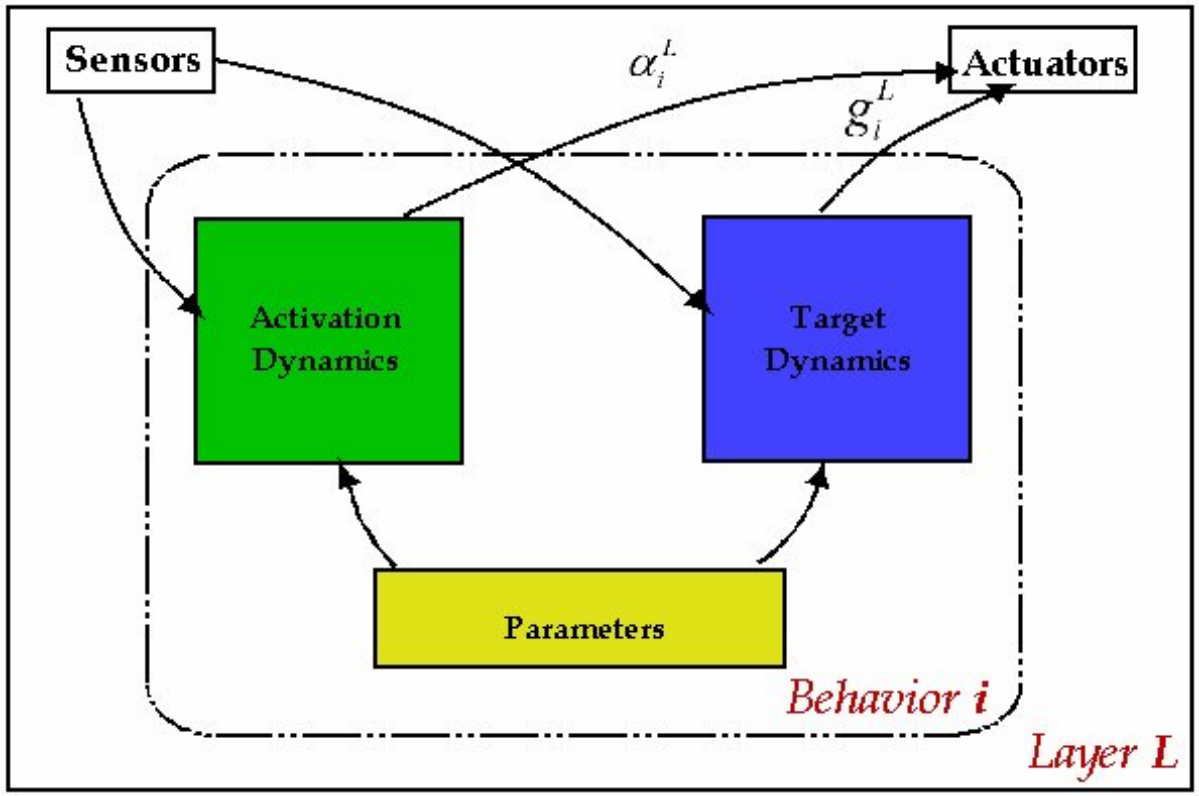


Dual Dynamics [Jäger]

- Aktivierungsdynamik
  - Wer darf etwas machen?
- Zieldynamik
  - Was macht man?
- Wirken auf **Aktoren**

Das Verhaltensmodell 4

Dynamik eines Verhaltens



Das Verhaltensmodell 3

- Sensordynamik
  - Sensoren auf verschiedene Ebenen
  - 3 Arten
    - Berechnete Sensoren
    - Aggregierte Sensoren
    - Mit Aktoren verknüpfte Sensoren
- Kommunikation zwischen Ebenen nur durch **Sensoren und Aktoren**!

MAAT

- **M**ulti **A**gent **A**uthoring **T**ool

  aber auch: altägyptische Göttin der Ordnung

- Grafischer Programmierrahmen für die FU Fighters Verhaltenssteuerung

Motivation

- Probleme des Verhaltenssystems:
  - Sehr gross (etwa **73 000** lines of code ohne Kommentare)
  - Schwierige, langsame und fehleranfällige Integration von neuen Elementen (Sensoren, Aktoren, Verhalten)
  - Unübersichtlichkeit
  - Verhaltensarchitektur und –konzept unsichtbar

Ziele des Frameworks

- Code Management
- Grafische Tools
- Automatisierung von Standardprozessen
- Das System “kennt” die Architektur und vermeidet Fehler
- Zugriff nur auf relevantes Code
- Leichte nachträgliche Integration von Modulen

Code Management

```cpp
// This is the implementation of the class TestMouseCtrl
#include "StdAfx.h"
namespace RobotOmniVerhalten {
TestMouseCtrl::TestMouseCtrl(RobotOmniEbene1* e, SpielEbene1
:       TestVerhalten("TestMouseCtrl", vm)
{
        this->e = e;
        this->s = s;
        init();
}
TestMouseCtrl::~TestMouseCtrl()
{
        end();
}
void TestMouseCtrl::zielfunktion()
{
e->a_zielPos.change( this, s->s_mouseZielPos.get());
e->a_homePos.change( this, Vec2f(0.f, 0.f));
e->a_isTorwart.change( this, false);
e->a_zielGeschw.change( this, 0.f);
//e->a_maxGeschw.change( this, 1.f);
e->a_fahrtausrichten.change( this, 1.f);
e->a_fahrtAusrichtenPos.change( this, s->s_mouseAusrichtenPo
e->a_zumBall.change( this, true);
e->a_ballausweichen.change( this, false);
}
void TestMouseCtrl::init()
{
}
void TestMouseCtrl::end()
{
}
}
```

.CPP

```cpp
#if !defined(maat_RobotOmniVerhaltenTESTMOUSECTRL_H_)
#define maat_RobotOmniVerhaltenTESTMOUSECTRL_H_

namespace RobotOmniVerhalten{
class TestMouseCtrl : public TestVerhalten
{
public:
        TestMouseCtrl(RobotOmniEbene1*  e, SpielEbene1* s,
        virtual ~TestMouseCtrl();

        RobotOmniEbene1* e;
        SpielEbene1* s;

        // Functions
        void zielfunktion();
        void init();
        void end();

};
}
#endif
```

.H

Code Management

```
1 // This is the implementation of the class TestMouseCtrl
2 #include "StdAfx.h"
3 namespace RobotOmniVerhalten {
4 TestMouseCtrl::TestMouseCtrl(RobotOmniEbene1* e, SpielEbene1
5 :       TestVerhalten("TestMouseCtrl", vm)
6 {
7         this->e = e;
8         this->s = s;
9         init();
10 }
11 TestMouseCtrl::~TestMouseCtrl()
12 {
13         end();
14 }
15 void TestMouseCtrl::zielfunktion()
16 {
17 e->a_zielPos.change( this, s->s_mouseZielPos.get());
18 e->a_homePos.change( this, Vec2f(0.f, 0.f));
19 e->a_isTorwart.change( this, false);
20 e->a_zielGeschw.change( this, 0.f);
21 //e->a_maxGeschw.change( this, 1.f);
22 e->a_fahrtausrichten.change( this, 1.f);
23 e->a_fahrtAusrichtenPos.change( this, s->s_mouseAusrichtenPo
24 e->a_zumBall.change( this, true);
25 e->a_ballausweichen.change( this, false);
26 }
27 void TestMouseCtrl::init()
28 {
29 }
30 void TestMouseCtrl::end()
31 {
32 }
33 }
```

```
1 #if !defined(maat_RobotOmniVerhaltenTESTMOUSECTRL_H_)
2 #define maat_RobotOmniVerhaltenTESTMOUSECTRL_H_
3
4 namespace RobotOmniVerhalten{
5 class TestMouseCtrl : public TestVerhalten
6 {
7 public:
8         TestMouseCtrl(RobotOmniEbene1*  e, SpielEbene1* s,
9         virtual ~TestMouseCtrl();
10
11         RobotOmniEbene1* e;
12         SpielEbene1* s;
13
14         // Functions
15         void zielfunktion();
16         void init();
17         void end();
18
19 };
20 }
21 #endif
```

Automatisch verwaltetes Code

Demonstration