Quick start tutorial for programming a SimSpark agent with the framework RoboNewbie_1.0

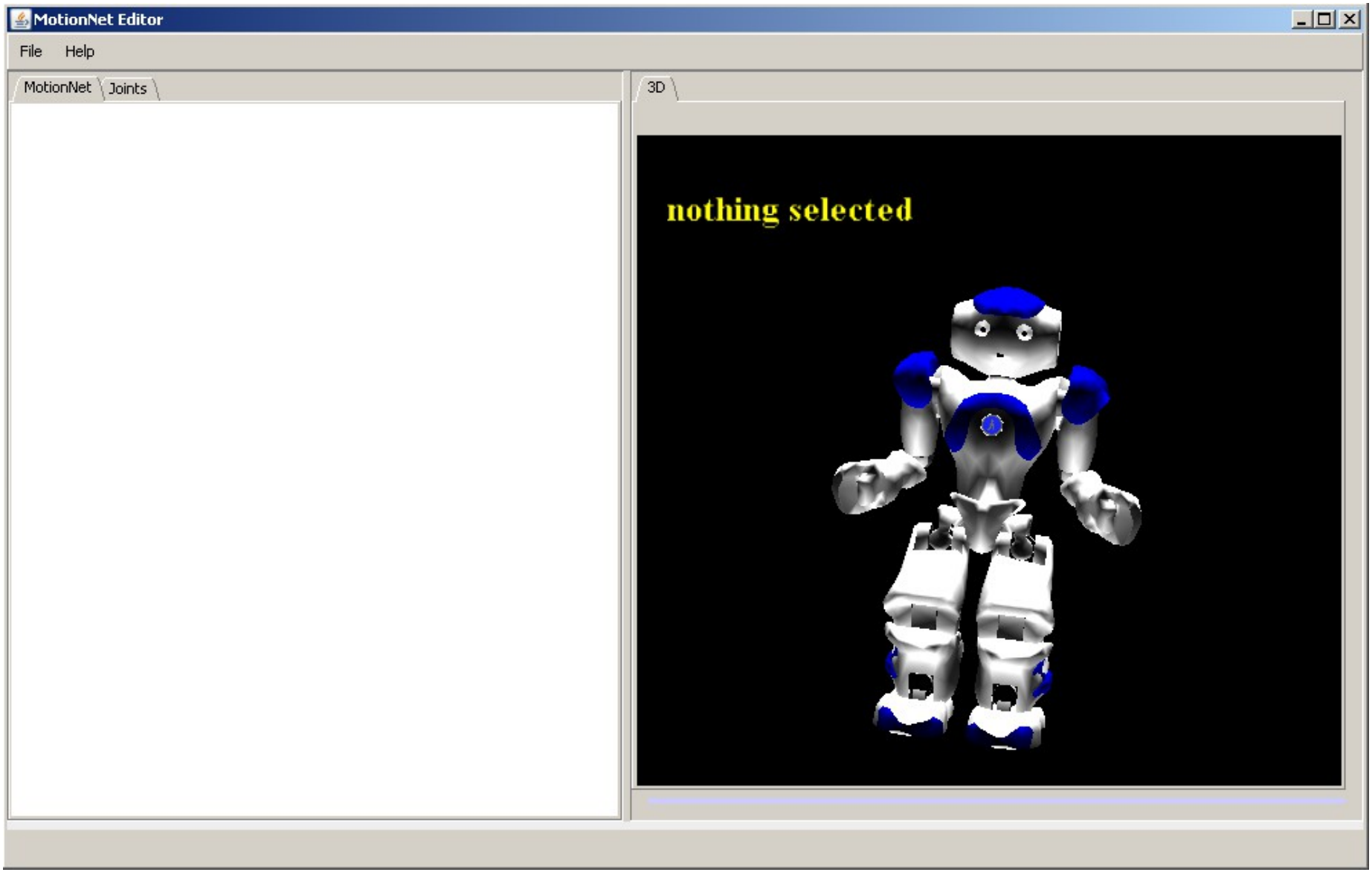
Monika Domanska, Hans-Dieter Burkhard
Institute of Informatics
Humboldt University Berlin, 10099 Berlin

hdb@informatik.hu-berlin.de
domanska@informatik.hu-berlin.defjs

14.6.2013

Check the installation

MotionEditor, started with **MotionEditor.jar**:



Check the installation

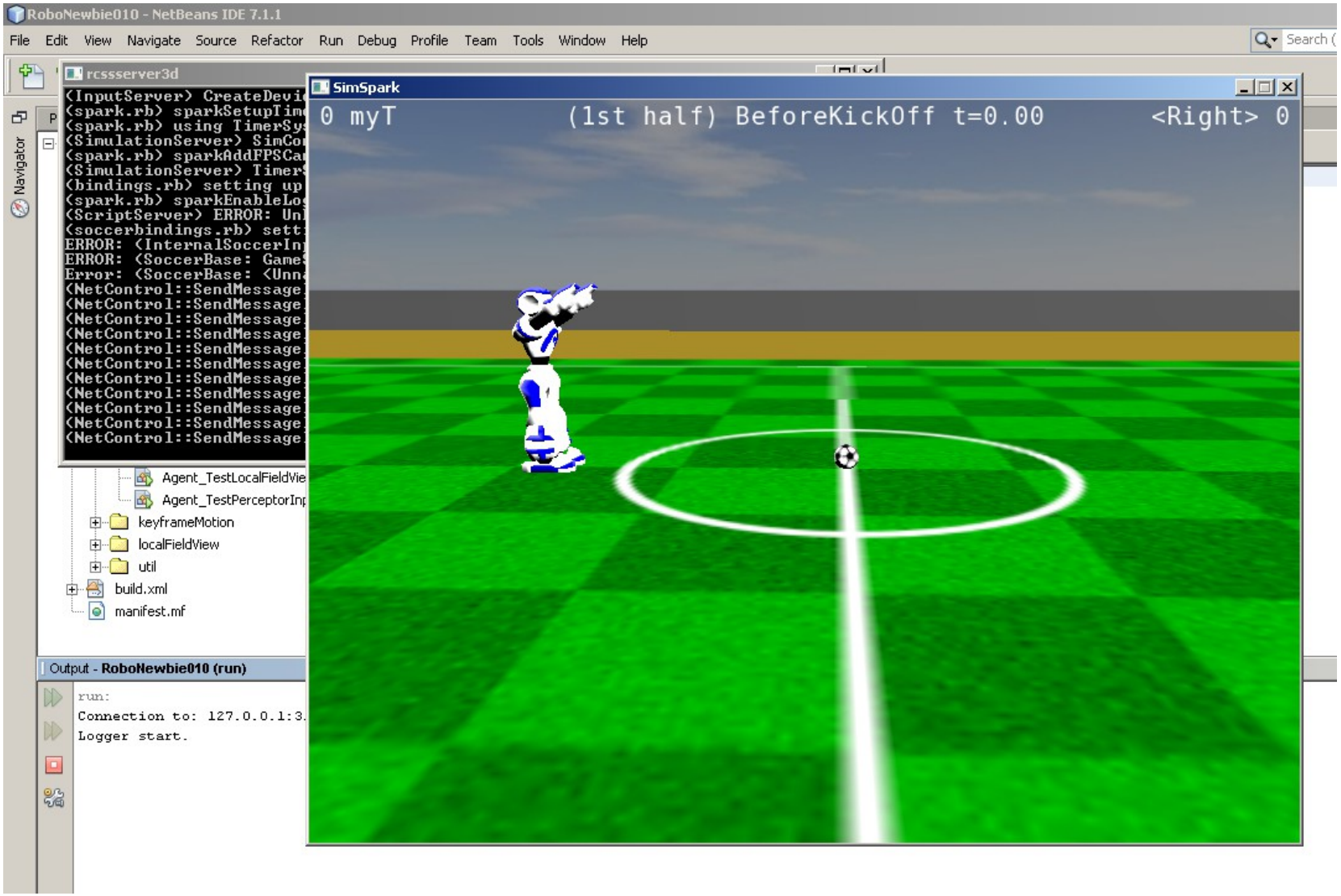
SimSpark and RoboNewbie:

1) Start SimSpark with **rcsserver3d.exe**.
2) Start NetBeans and open the RoboNewbie project in it.
   NetBeans >> File >> Open Project...
   >> [choose the unzipped RoboNewbie directory]
3) Go to the package "**examples**", and run
   Agent_BasicStructure.java
   (right click on the file name → Run File) .

In the SimSpark monitor window you should see a robot, after a moment it lifts the arms, then it stands still for a few seconds, and at the end the robot disappears.

Check the installation

SimSpark and RoboNewbie:



Outline

I. **Structure overview**

II. Architecture of agents

III. Classes for using perceptors

IV. Debugging agent code

V. Predefined effector usage: class KeyframeMotion

VI. Example to improve: agentSimpleSoccer

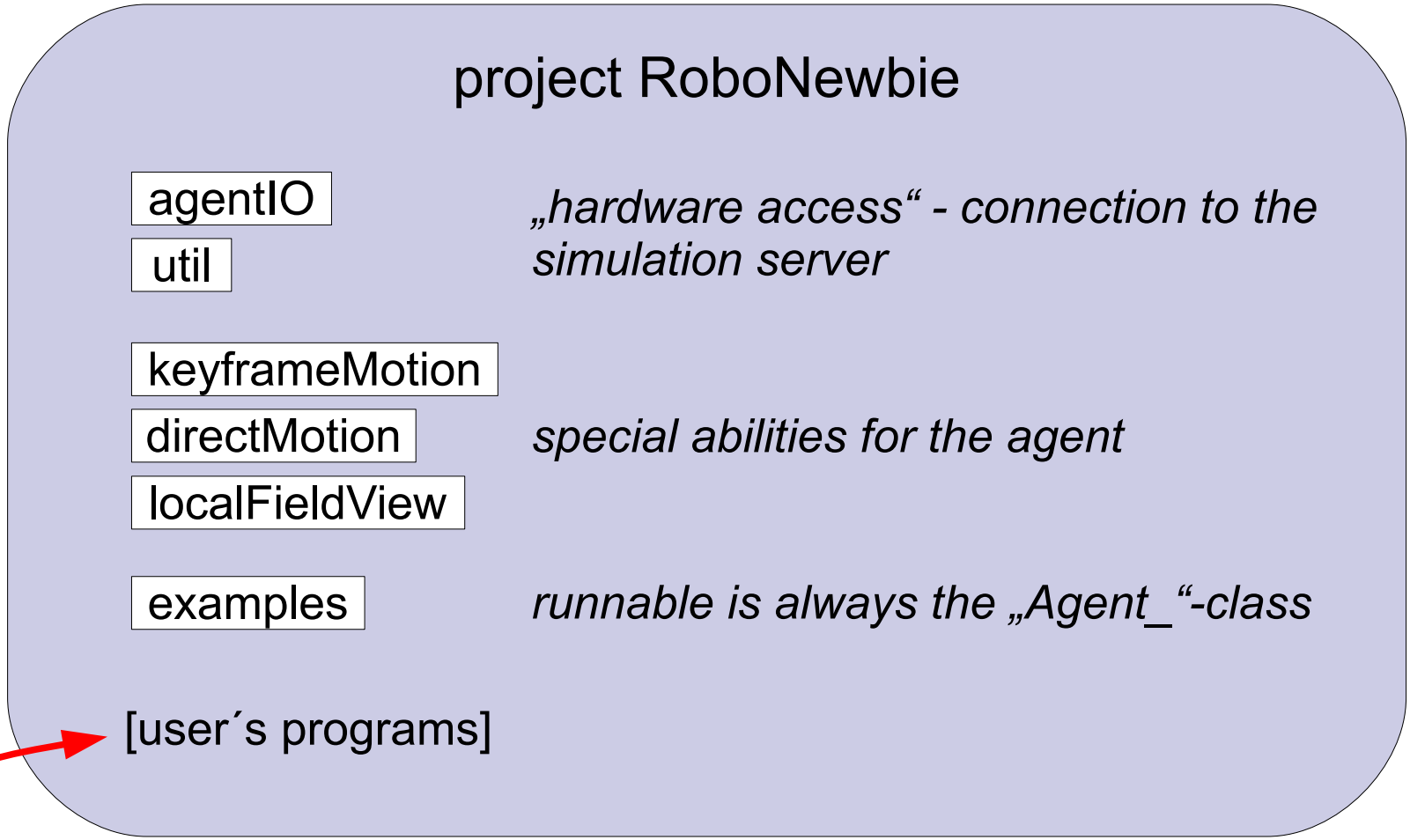
Structure overview

RoboNewbie:

- Java framework for programming the control of a SimSpark robot
- set of useful classes and data
- set of implementation examples
- distributed as a zipped NetBeans project
  - sources are in subdirectory „src“
  - documentation in Javadoc-comments inside the source code, read first the comments above the class definitions
  - data in subdirectory „keyframes“ necessary for running examples
  - used library in subdirectory „lib-apache-commons-math“
- Informations and software available at the RoboNewbie homepage: http://www.naoteamhumboldt.de/projects/RoboNewbie/

Structure overview

Java package structure:



Create your own programs here inside the RoboNewbie project (e.g. in a new package „myAgents“), because of data and library dependencies.

Outline

I. Structure overview

II. Architecture of agents

III. Classes for using perceptors

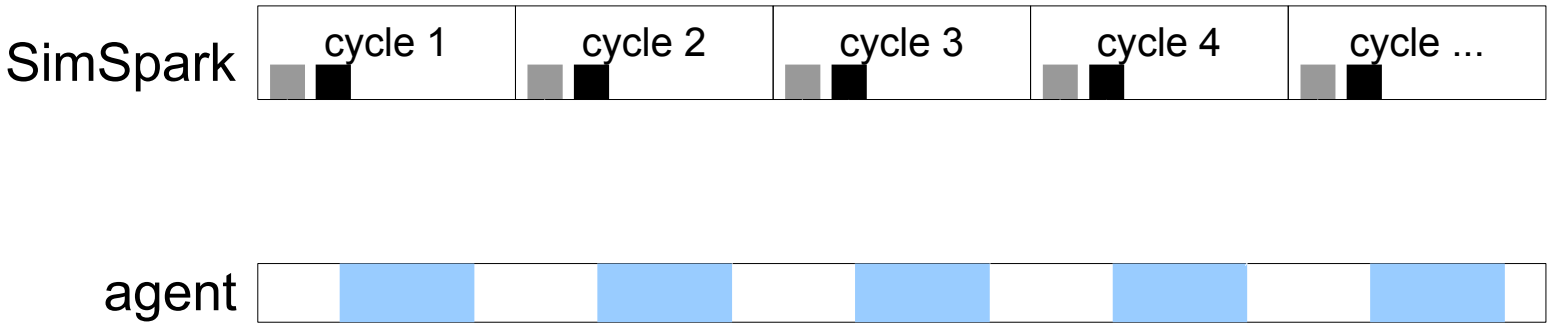
IV. Debugging agent code

V. Predefined effector usage: class KeyframeMotion

VI. Example to improve: agentSimpleSoccer

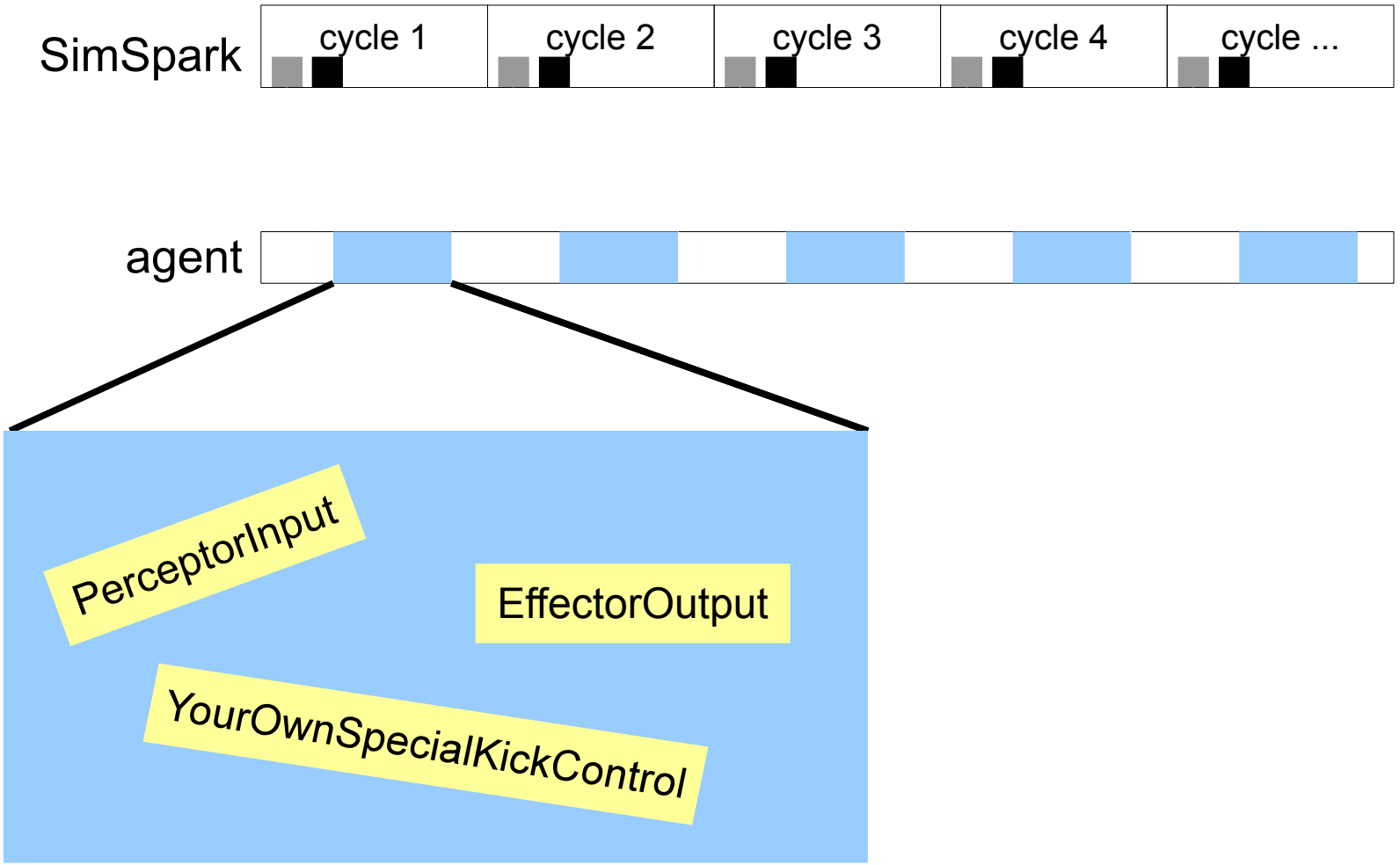
Architecture of agents

Components of a simulation:

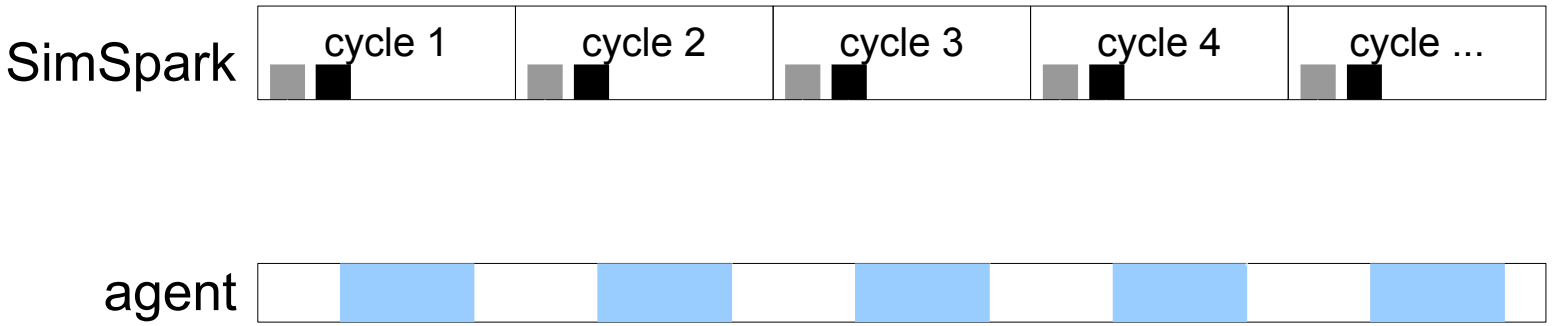


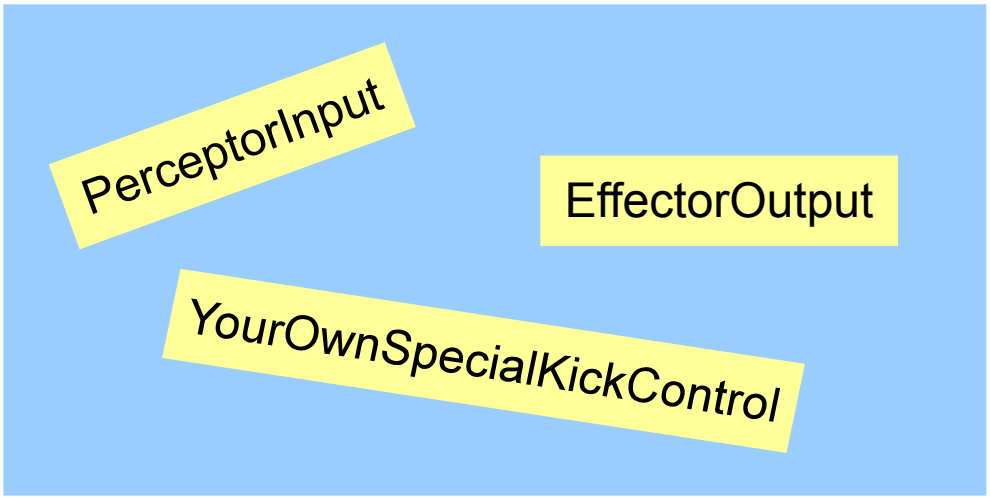
Architecture of agents

Components of a simulation:



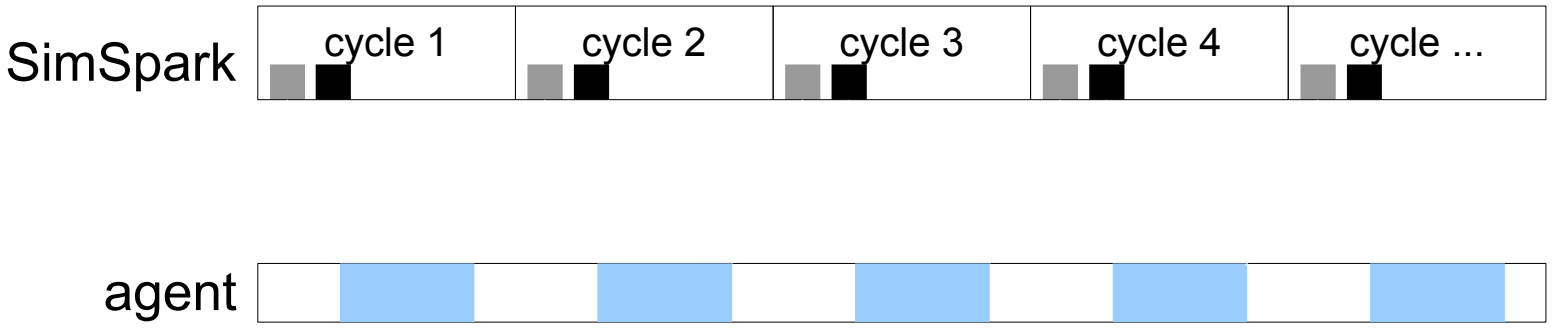
Architecture of agents

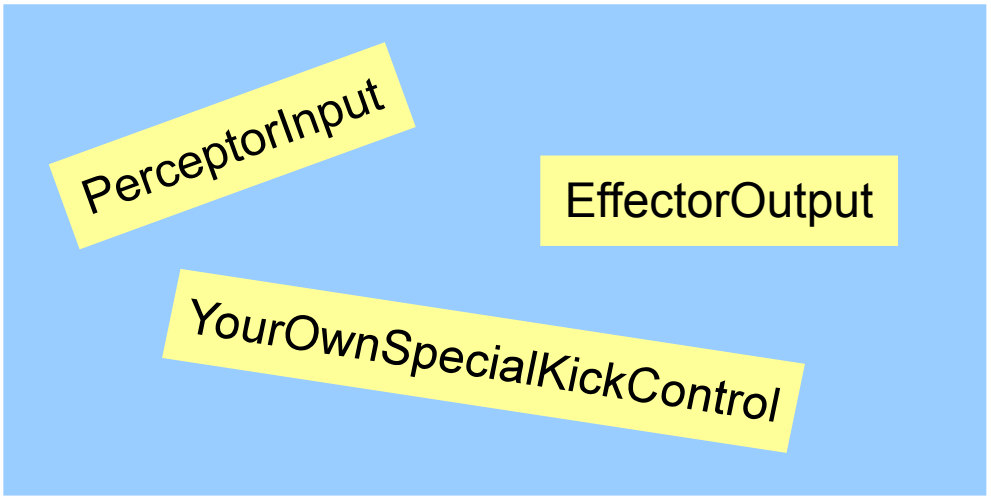




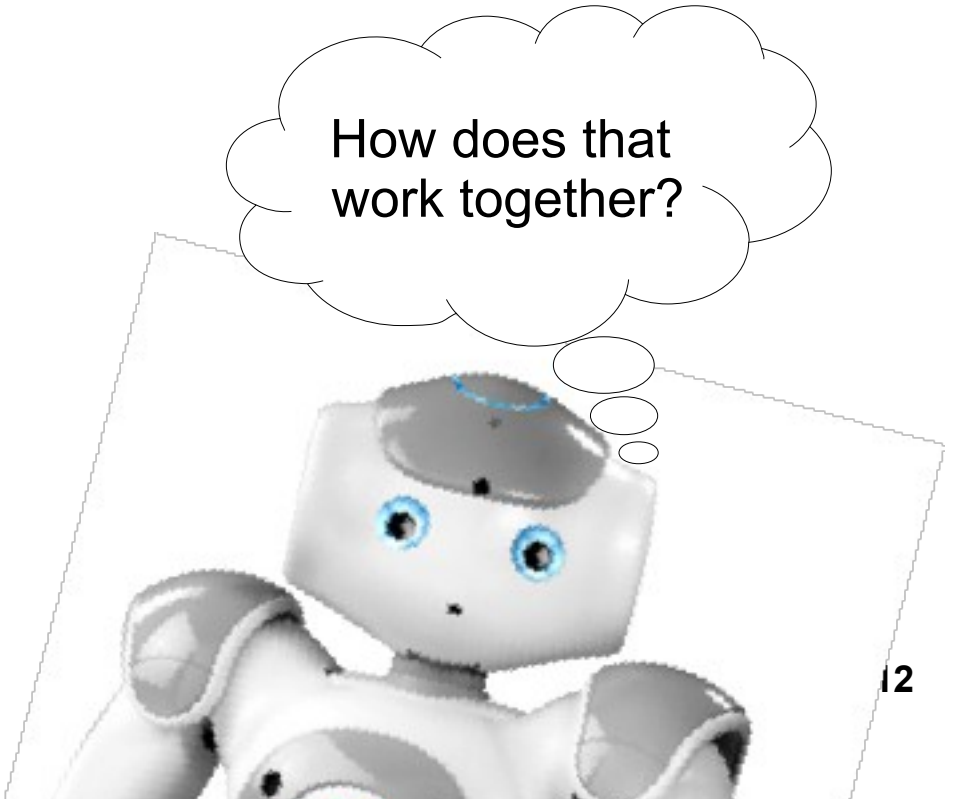
agent classes

Architecture of agents

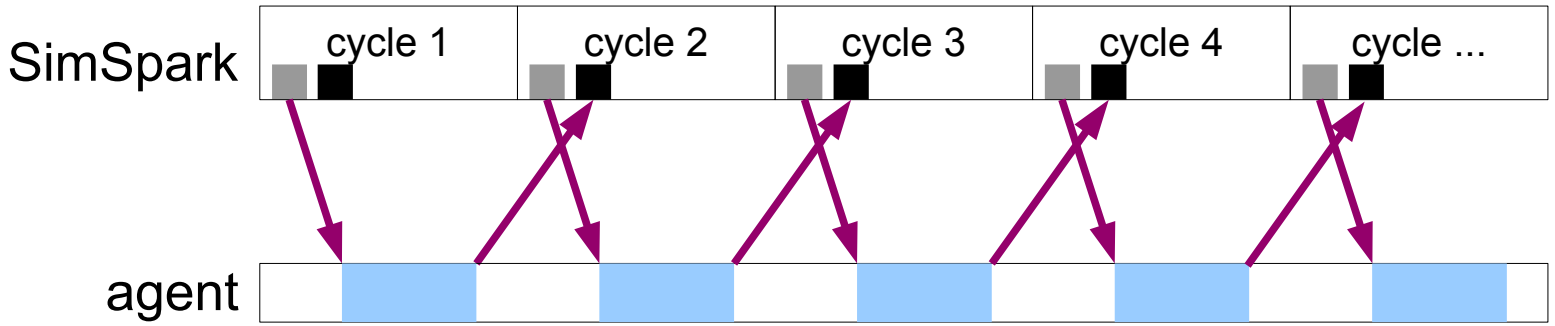




agent classes



Architecture of agents



1. PerceptorInput
2. YourOwnSpecialKickControl
3. EffectorOutput

„Agent_...“-class:
runnable,
synchronization of server and agent,
coordination of agent classes

agent classes

Architecture of agents

Synchronization with the SimSpark server:

Receiving a server message with perceptor values
and
sending an agent message with effector commands
in every server cycle.

Coordination of agent classes:

Defining the order of class methods depends on classes choosen for the agent.

Example:

Agent_BasicStructure.java

Architecture of agents

Exercise 1: Try out changing Agent_BasicStructure.

1) Open the RoboNewbie-Project in NetBeans.
   Make a new package for your own agents.
   Copy Agent_BasicStructure from package examples into your own package.
2) Start SimSpark with
   "[SimSpark root directory]/rcssserver3d.exe"
   Navigate at the monitor window with left mouse button, arrow keys, keys a,s,d,w, page up, page down.
3) Run your class Agent_BasicStructure to test if everything works
   (in NetBeans: right click on the filename → Run File).
4) Try changes:
   - ➔ Choose another initial position for the robot.
   - ➔ Change the effector commands in method run().
     Try out other velocities, they range from -2π to 2π.
     Try using other robot joints, you find the joint names in class RobotConsts in package util.

Outline

I. Structure overview

II. Architecture of agents

III. Classes for using perceptors

IV. Debugging agent code

V. Predefined effector usage: class KeyframeMotion

VI. Example to improve: agentSimpleSoccer

Classes for using perceptors

Three kinds of perceptors in SimSpark:

1.
for the server and game state:
Time,
GameState

2.
proprioceptive:
HingeJoint,
Accelerometer,
GyroRate,
ForceResistance

3.
for the simulated external environment:
Vision (objects like the ball, other players, …),
Hear

Classes for using perceptors

Classes receiving messages from the server and providing data:

1.	2.	3.
for the server and game state: Time, GameState	proprioceptive: HingeJoint, Accelerometer, GyroRate, ForceResistance	for the simulated external environment: Vision (objects like the ball, other players, …), Hear

received by class PerceptorInput	
in every server cycle	in every second (Hear) or third (Vision) server cycle
access provided only by PerceptorInput	more comfortable access to Vision provided by class LocalFieldView

Classes for using perceptors

Examples:

- For PerceptorInput:
  examples.Agent_TestPerceptorInput
  Note: PerceptorInput is already used in the Exercise 1 with Agent_BasicStructure.
- For LocalFieldView:
  examples.agentTestLocalFieldView.Agent_TestLocalFieldView
  We will use it in Exercise 2.

Classes for using perceptors

PerceptorInput → LocalFieldView

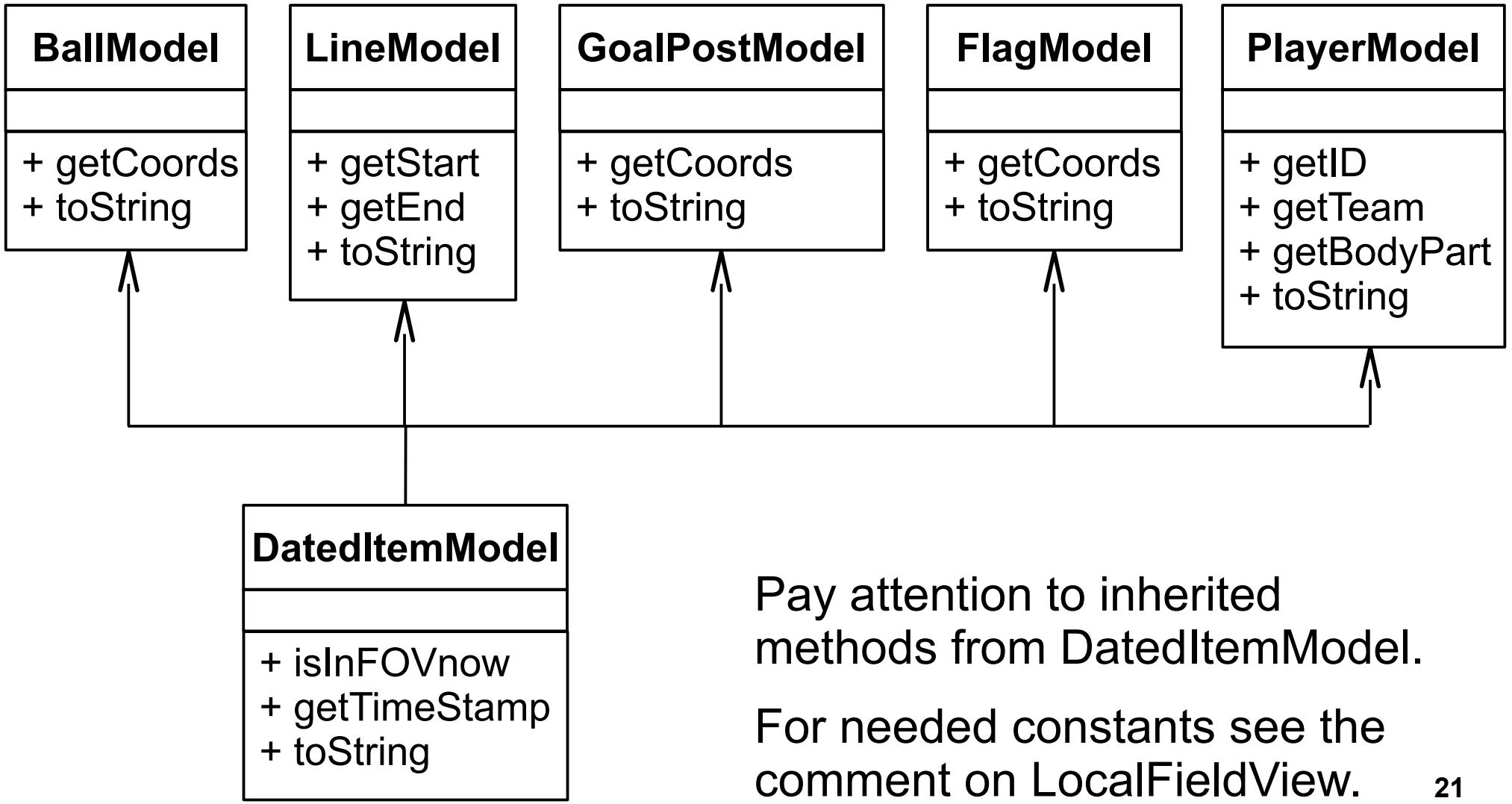
raw Vision-data

model („beliefs“) of actual field situation:

- local coordinate system
- coordinates of **field items** (ball, lines, goal posts, flags and other robots) and time of last sensing

Classes for using perceptors

Simplified class diagram for models delivered by LocalFieldView:



Pay attention to inherited methods from DatedItemModel.

For needed constants see the comment on LocalFieldView.

Classes for using perceptors

Exercise 2: Show, where the other agent is.

Implement an agent, which lifts the robots arm, when it senses another robot and moves the arm down, when it does not sense any robot. If the other robot is on the left side of your own one, lift the left arm, and the right arm for the right side.
(Limits for lifting and droping the arms are not important, just set the effectors to move into the correct direction.)

Instructions:
Change a copy of Agent_TestLocalFieldView. Define:

```
id = „2“
team = „simpleSoccer“
Beam coordinates: X=-1, Y=-1, Rot=90.
```

Use Agent_SimpleSoccer as the target. Start it at different initial positions. Some restrictions apply according to soccer rules.
Press „k“ (kick-off) and „b“ (drop ball) in the Monitor window.

Outline

I. Structure overview

II. Architecture of agents

III. Classes for using perceptors

IV. Debugging agent code

V. Predefined effector usage: class KeyframeMotion

VI. Example to improve: agentSimpleSoccer

Debugging agent code

Problem:

Using debug messages printed on *System.out* need too much time, and the agent can not synchronize with the 20ms-cycle of the SimSpark server.
This causes different strange behaviours of the controled robot, e.g. it does not execute motor commands as expected.

Solutions:

1. Use the class util.Logger and print the debug output after the agent program has finished. The usage of util.Logger is shown in the examples (same as for perceptor usage above):
   - examples.Agent_TestPerceptorInput
   - examples.agentTestLocalFieldView.Agent_TestLocalFieldView

2. Use the „Agent synchronized mode“ („Sync-mode“) of the SimSpark server and print the output during the runtime of your agent program.

Debugging agent code

„Agent synchronized mode“ (Sync-mode):
The SimSpark server does not run in real time – it waits in each cycle until it has received an agent message from every connected agent, and then it starts the next cycle.

> *System.out* can be used :)

How to set the Sync-mode:
in file „[SimSpark root dir]/spark.rb“ change line 46 to :

```
$agentSyncMode = true
```

The integrated monitor will stop together with the server, to see the simulation start another monitor window with „[SimSpark root dir]/rcssmonitor3d.exe“

Outline

I. Structure overview

II. Architecture of agents

III. Classes for using perceptors

IV. Debugging agent code

V. Predefined effector usage: class KeyframeMotion

VI. Example to improve: agentSimpleSoccer

Predefined effector usage: class KeyframeMotion

Keyframe motions: Whole motions, e.g. walking two steps, modeled in a sequence of postures „like in a comic“.

Class KeyframeMotion provides predefined motions:

- walk two steps forward
- stop walking
- big and small steps for turning right and left
- side steps to the left and right
- stand up from lying on the back
- roll over on the back (from lying on the front side)
- falling to the front and back
- motion the head down, left and right
- wave with both arms

Motions defined in .txt-files in „[RoboNewbie root directory]/keyframes“

Predefined effector usage: class KeyframeMotion

Usage of the motions:

1) Once: if the robot is ready for the next motion (i.e. if it has finished the last one), set the desired motion
   in any class, usually in a class representing the thinking

   AND

2) Continuously in every server cycle: call method executeKeyframeSequence()
   in method act() of the „Agent_“-class

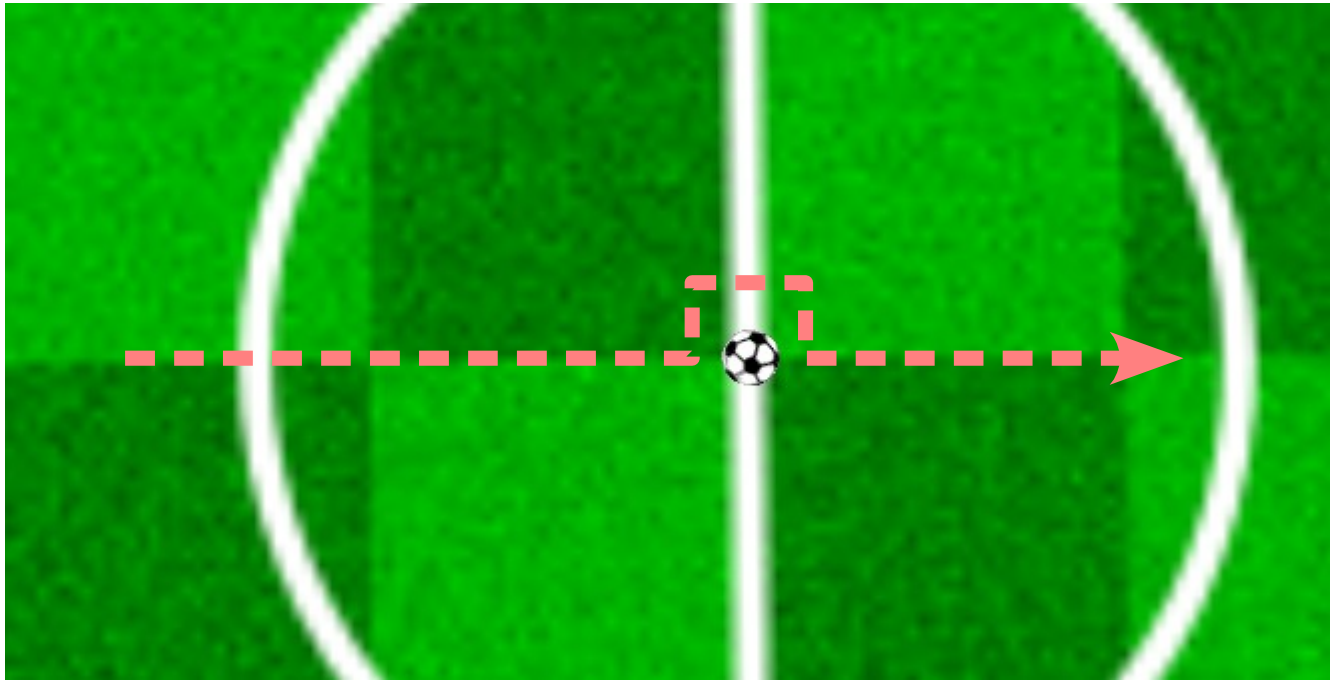
Example:

Agent_SimpleWalkToBall

Predefined effector usage: class KeyframeMotion

Exercise 3: Walk avoiding an obstacle.

Start the robot facing the ball. It should walk forward, not hitting the ball, but staying close to the direct way:



Predefined effector usage: class KeyframeMotion

Exercise 3: Walk avoiding an obstacle.

Instructions:
Change a copy of example agentSimpleWalkToBall. Define:

```
Beam coordinates: X=-1.5, Y=0, Rot=0.
```

(ID and team freely choosen.)
Use the ball model just to consider, when the robot should start evading. The evading itself should be a sequence of keyframe motions, not depending on the perceptors anymore.

Predefined effector usage: class KeyframeMotion

Two possibilities for designing new keyframe motions:

1. with a text editor

- each line represents one frame
- first value is transition time
- then values for target angles for the joints in the order as defined in RoboNewbie class util.RobotConsts

2. with program MotionEditor

- made by RoboCup team NaoTH from Humboldt University Berlin
- download executable and usage instructions from the RoboNewbie homepage

Predefined effector usage: class KeyframeMotion

Exercise 4: Design a motion for kicking the ball and try it out.

Instructions:

- Design the motion with the MotionEditor and save it in file „[RoboNewbie root directory]/keyframes/test.txt“.
  (Optional: You can generate the motion for the other side with examples.agentKeyframeDeveloper.OtherSideGenerator.java)
- Test the new motion with RoboNewbie example agentKeyframeDeveloper (see comment on class for usage).
- Integrate the motion to be provided by class KeyframeMotion as described in the comment on the class KeyframeMotion.
- Write an agent using the new motion to check, if it is integrated correctly.

Outline

I. Structure overview

II. Architecture of agents

III. Classes for using perceptors

IV. Debugging agent code

V. Predefined effector usage: class KeyframeMotion

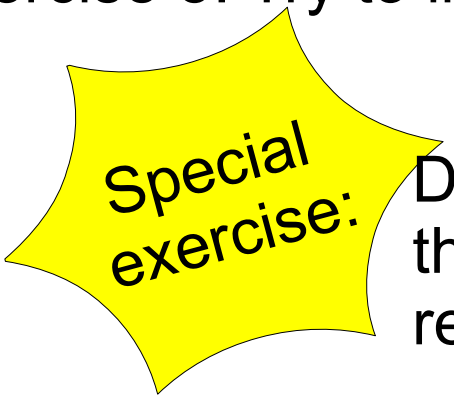
VI. Example to improve: agentSimpleSoccer

Example to improve: agentSimpleSoccer

- Uses all possibilities of RoboNewbie.
- Applies KeyframeMotion for the body motions
  and
  special motion class LookAroundMotion in package directMotion for the cyclic head motion.

  ↳ Motion classes are coordinated in „Agent_“-class.
- Needs 10 minutes to push the ball almost into the opponent goal.

Exercise 5: Try to improve agentSimpleSoccer.



Design or implement new motions and send them to us! We´re always looking for projects realized with RoboNewbie.

Thanks for your attention!